STUDIES ON DISCRETE PROGRAMMING(Dissertation_全文)

AUTHOR(S):

Ishii, Hiroaki

STUDIES
OF
DISCRETE PROGRAMMING

HIROAKI ISHII

June 1978

STUDIES
ON
DISCRETE PROGRAMMING

HIROAKI ISHII

June
1978

STUDIES
ON
DISCRETE PROGRAMMING

by

Hiroaki Ishii

Submitted in partial fulfilment of the

requirement for the degree of

DOCTOR OF ENGINEERING

(Applied Mathematics and Physics)

at

Kyoto University

Kyoto Japan

June

1978

PREFACE

A considerable portion of mathematical programming problems found in the various engineering fields are closely related to discrete quantities. These problems can generally be formulated as discrete programming problems where a given objective function is to be optimized under certain conditions, one of which is to restrict some variables to discrete values. Among them, there still remain many important but unsolved problems in practical sense.

The purpose of this dissertation is twofold; first to investigate generalized versions of some well known problems, such as the assignment problem, the traveling salesman problem and the knapsack problem, and secondly to propose new practically efficient algorithms for some known problems such as the integer fractional programming problem and the quadratic fractional programming problem. To begin with, this dissertation generalizes the ordinary assignment problem toward a new direction in that its generalization enables us to incorporate network type constraints into the assignment problem. Moreover, an efficient algorithm, based on the branch-and-bound method and the decomposition principle of Dantzig-Wolfe's type, is also developed for the problem. Next, the dissertation treats a delivery route problem of a production system. This problem provides a minimax type traveling salesman problem with a minimax type objective function. Its objective function can be more general than that of the ordinary

traveling salesman problem in the sense that the former includes the latter as a special case. The problem is decomposed into subproblems and this decomposition enables us to develop an algorithm based on dynamic programming procedure. The dissertation further generalizes the problem by introducing the capacity constraint into the minimax type traveling salesman problem. Though this capacity constraint makes it difficult for us to find feasible solutions, a decomposition of the problem into subproblems enables us to develop another algorithm based on dynamic programming problem. Subsequently, the dissertation investigates fractional programming problems, the integer fractional programming problem and the fractional knapsack problem. A primal cutting plane algorithm is tailored for the integer fractional programming problem. On the other hand, the fractional knapsack problem is a generalized knapsack problem with a linear fractional objective function. Together with several useful properties of the problem, the dissertation derives an efficient algorithm fully utilizing greedy solutions of both the fractional knapsack problem and the ordinary knapsack problem. The proved properties include an upper bound of the number of the subsidiary problems to be solved and sensitivity analysis with respect to the right hand of the inequality constraint. Finally, the dissertation gives two algorithms for the quadratic fractional programming problem. One is based on the parametric programming procedure and the other is a modification of Dinkelbach's approach for the general fractional program-

ming problem. Further, the dissertation notes that appropriate methods from numerical analysis for finding zero points of functions facilitate finding efficient algorithms for the fractional programming problem.

ACKNOWLEDGEMENT

The author wishes to express his sincere appreciation to Professor Hisashi Mine of Kyoto University for supervising this dissertation. He awakened author's interest in Discrete Programming.

The author is also indebted to Professor Toshio Nishida of Osaka University for his great help and suggestions in the completion process of the dissertation.

The author wishes to thank Associate Professor Toshihide Ibaraki of Kyoto University for his continual suggestions and contributions to the content and organization of the dissertation. Without his help, this dissertation would perhaps not be completed. His thanks should also be extended to Professor Toshiharu Hasegawa, and Associate Professor Katsuhisa Ohno of Kyoto University , and Associate Professor Yoshio Tabata of Osaka University for their invaluable comments and criticisms.

Finally the author is grateful of his colleagues in Mine's Laboratory of Department of Applied Mathematics and Physics, Kyoto University, for their comments and freindship while the author was a student therein.

CONTENTS

CHAPTER 1 INTRODUCTION

1.1 Purpose of the Dissertation

This dissertation treats discrete programming which is an important subclass of mathematical programming. Discrete programming aims to optimize a given objective function under given constraints, with respect to its variables, some of which are restricted to take only discrete values.

A variety of discrete programming problems have been encountered in many fields such as system sciences, operations research and information sciences, among others. The area to which discrete programming is applied has been growing at amazing rate. Listed below are examples of typical and well-studied problems included in the field of discrete programming.

(a) Graph or Network —— matching problem, traveling salesman problems, colouring problems, critical path problems, shortest path problems, flow problems, etc.

(b) Sequencing —— flow shop scheduling problems, job shop scheduling problems, line balancing problems, etc.

(c) Distribution —— assignment problems, transportation problems, transshipment problems, facility location problems, fixed charge problems, etc.

(d) Packing —— knapsack problems, bin packing problems, cutting problems, etc.

It should be first noted that these discrete programming problems usually have finitely many feasible solutions. Hence, enumeration can be in principle applied to find an

optimal solution among all feasible solutions. However the enumeration of feasible solutions, if done exhaustively, requires prohibitively large computational time, and therefore even moderate size problems often exceed the capacity of computers. This implies that some ingenious ideas are necessary to avoid the total enumeration, in order for a solution method to be practical. However, in spite of the efforts of many creative people, no method has been, is, and perhaps will be discovered which is uniformly effective for all discrete programming problems. In this sense, special solution procedure is required for each given problem so that the structure of a given problem is fully exploited to yield an algorithm which is efficient enough to be practical.

An aim of this dissertation is to provide efficient algorithms for some discrete programming problems and to extend the scope of some known algorithms developed for some specific problems in the sense that more general classes of problems can be solved by them.

1.2 Historical Background

This section gives defnition and brief historical review of some problems discussed in discrete problem, restricting to those closely related to the problems investigated in this dissertation, i.e., the assignment problem, the traveling salesman problem, the knapsack problem and several programming problems with fractional objective functions.

In 1955, H.W.Kuhn proposed the well known Hungarian Method [Kuhn,'55] for the assignment problem which is formulated as follows:

$$\text{Minimize} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$\text{subject to} \sum_{i=1}^{n} x_{ij} = 1, \quad j=1,\ldots,n,$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i=1,\ldots,n,$$

$$x_{ij} = 0 \text{ or } 1 \quad \begin{pmatrix} i=1,\ldots,n \\ j=1,\ldots,n. \end{pmatrix}$$

Originally, the assignment problem has the following actual meaning. Given the assignment cost c_{ij} of assigning i-th person to j-th job for i=1,...,n, j=1,...,n, how we should assign n persons to n jobs in order to minimize the sum of assignment cost under the condition that one and only one person is to be assigned to each job.

His method was perhaps the first efficient algorithm

for discrete programming, not to speak of the assignment problem. Refining Kuhn's idea, J.Munkres [Munkres,'57] also proposed an algorithm for the assignment problem and transportation problem. As is stated in Chapter 2, since then, several algorithms for the assignment problem and variants of it have been proposed. This dissertation also investigates a variant of it in Chapter 2.

On the other hand, a similar but slightly different problem, called the traveling salesman problem has been notorious for defying any efficient algorithm except for a few special cases. As the name "traveling salesman problem" indicates, the original problem is as follows: A salesman is required to travel through n cities by visiting each city exactly once to sell a certain commodity and return to the initial city. Given travel time t_{ij} from city i to city j ($i \neq j$), find the order of cities to be visited, so that the total travel time is minimized.

This problem is mathematically formulated as follows:

$$\text{Minimize} \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)}$$

$$\text{subject to} \quad \pi=(\pi(i)) \in \Pi_n,$$

where Π_n is permutation group of order n.

Many papers have been written on the problem. They include algorithms based on dynamic programming by M.Held and R.M.Karp [Held and Karp,'62], R.Bellman [Bellman,'62], and algorithms based on branch-and-bound method by J.D.C.

Little et.al. [Little et.al.,'63] and Held and Karp [Held and Karp,'70 and 71]. These algorithms, however, seem to require the computational time exponentially growing with respect to the problem size. This dissertation investigates two variants of it and proposes solution algorithms for them.

The knapsack problem stated below is an integer programming problem with a special property that only one constraint is imposed. This simply structured problem is another famous example of a problem which does not have any efficient algorithm. The knapsack problem is defined as follows: Find the most desirable set of items a camper should pack in his knapsack, given measure c_i of desirability of each item i and its weight a_i, under the constraint of the maximum weight b that the knapsack (or camper) can carry.

Mathematically , this problem is formulated as follows:

$$\text{Maximize} \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \sum_{j=1}^{n} a_j x_j \leq b,$$

$$x_j \geq 0, \text{ integer, } j=1,\ldots,n.$$

The name "knapsack problem" was apparently suggested by G.B.Dantzig [Dantzig,'57]. An alternative name "loading problem" was given by R.Bellman. Many advances in solution techniques for the knapsack problem have

followed the pioneering work by P.C.Gilmore and R.E.Gomory on the cutting stock problem ([Gilmore and Gomory,'61, '63, '65 and '66]). They used the knapsack problem as subproblems of the cutting stock problem. They proposed recursive procedures based on dynamic programming for solving the knapsack problem. Other dynamic programming procedures were also given in [Dantzig,'57] and [Bellman, '57]. Algorithms based on branch-and-bound method were proposed by P.C.Kolesar [Kolesar,'67], H.Greenberg [H.Greenberg,'69], H.Greenberg and R.L.Hegerich [Greenberg and Hegerich,'70] and V.A.Cabot [Cabot,'70]. Mathematical properties of optimal solutions of knapsack problems have been also intensively investigated. For example, the periodicity property was discussed in [Gilmore and Gomory, 66], [Shapiro,J.F., and H.M.Wagner,'67], [T.C.Hu,'69] and [Garfinkel,R.S., and G.L.Nemhauser,'72]. M.J.Magazine et.al. [M.J.Magazine et.al.,'75] proposed the greedy algorithm of the knapsack problem, which is heuristic but quite efficient, and gave a condition under which the greedy solution becomes an exact optimal solution. The dissertation fully utilizes above goodness of the greedy solution for investigating a generalized knapsack problem, or, a knapsack problem with a fractional objective function instead of the linear objective function.

Finally, we comment on the discrete programming problems with fractional objective functions, formally defined as follows;

$$\text{maximize} \quad N(x)/D(x)$$

$$\text{subject to} \quad x=(x_j) \in S \subseteq R^n, \quad x_j; \text{ integer}, \quad j=1,\ldots n,$$

where R^n denotes n-dimensional euclidian space.

Compared with the above mentioned problems, few papers have been written on this topic. Fractional objective function reflects certain situations that both cost and reward in the choice of optimal plans are to be considered, that is, cost is to be minimun while reward is to be maximum. Moreover, in a sense, the ordinary objective function is a special case of the fractional objective function. P.L.Hammer and S.Rudeanu discussed the fractional linear minimization problems with 0-1 variables in their book [Hammer and Rudeanu,'68]. P.Robillard [Robillard,'71] investigated the (0,1) hyperbolic programming problem (hyperbolic programming is often used to mean fractional programming). M.Gruspan [Gruspan,'73] proposed a general algorithm for the linear fractional programming problem with integer variables. The dissertation provides an algorithm for the integer fractional programming problem.

1.3 Brief Review of Related Areas

This section summarizes some results in the areas related to the subjects of discussion in the dissertation. These areas include integer programming, graph theory, dynamic programming and branch-and-bound method. The emphasis is put on solution methods because the basic ideas in these solution methods will be frequently used throughout the subsequent discussion.

A. Integer Programming

An integer programming problem is generally written as follows:

$$\text{Maximize} \quad f(x)$$

$$\text{subject to} \quad x=(x_j) \in S \subseteq R^n$$

$$x_j; \text{ integer}, \ j \in N_1=\{1,\ldots,n_1\},$$

where R^n is n-dimensional euclidian space and $n_1 \leq n$. In particular, a pure integer linear programming problem is written as follows:

$$\text{Maximize} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \ , \ i=1,\ldots,m,$$

$$x_j \geq 0, \text{ integer}, \ j=1,\ldots,n.$$

Moreover, if each x_j is restriced to be 0 or 1, the following 0-1 integer programming problem is derived;

$$\text{maximize} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i=1,\ldots,m,$$

$$x_j = 0 \text{ or } 1, \quad j=1,\ldots,n.$$

The concept of integer programming may date back to almost the same time as that of linear programming. However the importance of integer programming has not fully been realized until Dantzig emphasized its importance in his tutorial paper [Dantzig,'57 and '60]. Soon after, Gomory proposed an iterative algorithm with finite convergence property, well known today as cutting plane method ([Gomory,'58]). His method attracted great attention of researchers and various new integer programming algorithms followed.

Algorithms known today may be classified into three types; (1) cutting plane method (2) enumerative method (3) group theoretic method.

(1) *Cutting plane method*

This was originally proposed by Gomory in the paper [Gomory,'58] of 1958. He has shown that any integer linear programming problem can be solved by repeating the following procedure: Solve the problem as the ordinary linear programming one. If the obtained solution is not integer,

adjoin one constraint (this constraint is called a cut) which cuts off the current solution of linear programming problem, but still retaining all integer feasible solutions, and then reoptimize the modified linear programming problem. The above procedure is repeated until the optimal solution of the linear programming problem becomes an integer solution. Strictly speaking, his algorithm is called the dual cutting plane method for pure integer linear programming problems because it uses the dual simplex method to solve the generated linear programming problems.

The primal version of Gomory's algorithm was proposed by R.D.Young [Young,'65]. It is called the rudimentary primal cutting plane algorithm. This algorithm was later simplified by F.Glover [Glover,'68] and Young [Young,'68]. Young's simlified primal cutting plane algorithm is made mention of in Chapter 5 in connection with a new algorithm for the integer fractional programming problem.

Variants of additional constraints, cuts, have also been proposed. Among them, we mention here only intersection cut and convexity cut introduced by E.Bals [Balas,'71] and Glover [Glover,'69] respectively.

Recently, R.G.Jeroslow [Jeroslow,'74] attempted to unify from the theoretic view point the vast variety of the cutting plane methods proposed so far.

(2) *Enumerative method*

The principle of this approach is branch-and-bound method described later. It takes advantage of the finiteness of the set of feasible integer solutions.

Balas [Balas,'65], Glover [Glover,'65] and A.M. Geoffrion [Geoffrion,67a] (he called this method implicit enumeration) first applied the branch-and-bound computation to pure integer linear programming problems. Their methods are especially suited for 0-1 integer programming problems and seem to be one of the most efficient ones currently available.

(3) *Group theoretic method*

This method was also first proposed by Gomory [Gomory, '65]. He showed that by relaxing nonnegativity condition on certain variables and maintaining their integrality condition, a pure integer programming problem can be transformed to a linear minimization problem on a finite abelian (i.e., additive) group defined from the congruence relation modulo 1. Basic approach is again branch-and-bound method and utilizes the above mentioned linear minimization problem.

Some research has also been done to clarify the relationship between optimal solutions of this derived minimization problem and those of the original problem ([White,W.W.,'66], [Shapiro,J.F.,'68a and '68b], [Gorry, G.A. and J.F.Shapiro,'71]).

B. Graphs

A graph consists of a vertex set $V=\{v_1,v_2,\ldots,v_n\}$ and an edge set $E\subset V\times V$. An edge connects a vertex with another vertex. Usually, two vertex defining an edge are not ordered, that is, an edge does not have a direction. If each edge has a direction, such a graph is called a directed graph. Some numbers are often associated with

edges to represent their characteristics. In this case the graph with these numbers is called a network.

Graphs appear in various aspects of the formulation and analysis of discrete programming problems. For example, the traveling salesman problem can be represented as a graph where each vertex denotes a city and each edge represents a road connecting two cities. A feasible solution of the problem is given by a circular path that visits each vertex exactly once and returns to the starting vertex.

Graph theory was born in 1736 when L.Euler settled the famous unsolved problem of his days called the Königsberg bridge problem. That problem was to find a circular path that passes each edge exactly once and returns to the starting vertex. Subsequently, in 1847, G.R.Kirchhoff developed the basic concepts and theorems concerning trees in graphs from his investigation of electric networks. Also in 1857, Caley studied trees arising from organic chemical isomers. In 1859, W.R. Hamilton proposed a famous puzzle "Around the World". That problem was to find a hamiltonian circuit in a graph representing the world. After then, the celebrated "four color problem" came into prominence in the field of graph theory. This problem had remained as a conjecture for a long period until it was finally settled affirmatively by W.Haken and K.Appel in 1976. The various results on graph theory have been obtained by many mathematicians, operations researchers, among others. They can be found in the books, [Ford & Fulkerson,'62], [Berge,C.,'62 and '73],

[Harary,F.,'69] and [Christofides,N.,'75] and others.

H.Whitney [Whitney,'35] introduced "martoid" as a generalized concept of graphs, and W.Tutte [Tutte,65] developed it. Another generalization, "hypergraph", was also proposed by C.Berge [Berge,'73].

C. Dynamic programming

Dynamic programming is one of the powerful tools for optimization. The basic idea is to transform a given problem with many decision variables into an equal number of subproblems, each containing only one decision variable. This is possible if the problem satisfies a certain condition called "the principle of optimality" ([Bellman, '57]) which is stated as follows.

(*The principle of optimality*)

An optimal set of decisions has the property that whatever the first decision is, the remaining decisions must be optimal with respect to the outcome which results from the first decision.

Optimization problems with many variables are usually governed by the following rule of thumb: The computation requirement increases exponentially with the number of variables, but only linearly with the number of subproblems. Thus there can be great computational saving by formulating a problem with many variables as a set of such subproblems each of which has only one variable. Optimal decisions for the subproblems are calculated sequentially. The sequential calculation is the essence of dynamic programming. When all subproblems are solved, an optimal decision for the entire problem is constructed from the

optimal solutions of such subproblems. Certain discrete programming problems, such as the traveling salesman problem and the knapsack problem were solved by dynamic programming.

Dynamic programming has been apparently used ([Wald, A.,'50], for example) long before it was named by Bellman. Undoubtedly, however, Richard Bellman is the father of dynamic programming. His research on the dynamic programming at Rand Corporation in 1950's was culminated in his first book [Bellman,'57]. He has continued to be extremely prolific in his writing on the dynamic programming ([Bellman,R. and S.Dreyfus,'62], etc) . R.Aris investigated discrete cases [Aris,'64] and R.A.Howard clarified the relation between dynamic programming and Markov process [Howard,'60]. Nemhauser made explicit the role of a basic monotonicity property of cost functions in formulating the fundamental equations of dynamic programming ([Nemhauser,'66[) . M.E.Thomas [Thomas,'76] surveyed recent results of dynamic programming.

D Branch-and-bound method

The underlying idea of branch-and-bound method is to decompose a given problem into several partial problems of smaller size. The decomposition is repeatedly applied to the generated partial problems, until each decomposed problem is either solved or proved not to include an optimal solution of the original problem. This operation is called branching operation. Branching operation, if done completely, results in the enumeration of all possible solutions. Computational efficiency of this type of branch-

and-bound method may be prohibitively low.

In order to reduce the number of partial problems to be searched, some tests are applied to each generated partial problem. If some of the partial problems are fathomed by these tests, then they are excluded from further considerations. These tests require to compute upper bound and/or lower bound of each partial problem. A partial problem (minimization problem) is terminated if the obtained lower bound is not smaller than the best upper bound of the optimal value of the original problem currently available. This operation is called bounding operation.

Another important point in constructing efficient branch-and-bound algorithm is the searching order of the partial problems. Typical search strategies are depth-first serach (linear search or single-branch search) (e.g., [Agin,Y.,'66] [Geoffrion,A.M.,67] [Glover,'65] [Golomb,S.W. and L.D.Baumert,'65] [Lawler,E.L. and D.E. Wood,'66]), best bound search (or minimum value search) (e.g., [Agin,'66] [Balas,'67] [Lawler and Wood,'66] [Nilsson,N.J.,'71]), breadth-first search and heuristic search (e.g., [Hart,P.E.,et.al.,'68] [Nilsson,'71] [Ibaraki,T.,'76]).

The first two strategies are frequently used in practical applications for the following reasons. Depth-first search consumes a memory size which is only a linear function of the original problem size, and its implementation is relatively easy. On the other hand, best-bound search minimizes the number of partial problems decomposed prior

to termination and hence a high performance is expected. It is noted, however, that this tends to consume a memory size which unfortunately is a exponential function of the original problem size.

Little et.al. first used the name, "branch-and-bound", in their paper [Little et.al.,'63] to solve the traveling salesman problem. Since then, branch-and-bound method has been combined into many solution algorithms for various discrete programming problems. General treatment of the principle was discussed by many researchers: P.Bertier and Roy.'64 [P.Bertier and Roy,'64], Lawler and Wood [Lawler and Wood,'66], Y.Agin [Agin,'66], Balas [Balas,'68], L.G.Mitten [Mitten,'70], T.Ibaraki [Ibaraki, '76a, '76b, 77a and '77b] and others.

1.4 Outline of the Dissertation

The dissertation consists of eight chapters. Some chapters discuss generalizations of some known discrete programming problems and other chapters gives new algorithms for some discrete programming problems including those with fractional type objective functions.

In Chapter 2, a generalization of the assignment problem is proposed in the sense it is defined on a network rather than a bipartite graph. This generalized formulation allows us to include the technological order and constraint into the assignment problem. An algorithm based on branch-and-bound principle is given. It is shown that each partial problem can be efficiently tested by the linear programming decomposition approach. Specifically, a lower bound for a partial problem is computed by repeatedly solving the associated ordinary assignment problem and the critical path problem. Some computational results are also included. Chapter 3 proposes a new type of traveling salesman problem, which comes from the routing problem of the delivery of resources for production. The problem is different from the ordinary one in that it has a minimax type objective function, which makes the problem considerably more difficult. To avoid the complete enumeration, however, it is shown, that the original problem can be decomposed into a series of subproblems, each of which can be solved by dynamic programming approach. Optimal solutions of subproblems are then combined to obtain an optimal solution of the original problem. Chapter 4 proposes still another version of the traveling

salesman problem. The problem is the same as the original traveling salesman problem except that it has a capacity constraint. This capacity constraint is added to both of the ordinary traveling salesman problem and its minimax version discussed in Chapter 3. Again, solution algorithms based on dynamic programming applied in a different manner from the one in Chapter 3 are given for these problems.

Chapter 5-Chapter 7 deal with fractional type problems that often comes from the pursuit of effective investment. Chapter 5 discusses a linear fractional programming problem with integer variables. Combining Dinkelbach's approach for fractional programming problems and Young's primal cutting plane algorithm for integer programming problem, it is shown that only a slight modification is necessary to solve linear fractional programming problem with integer variables. Finite convergence of this algorithm is proved. Chapter 6 investigates a fractional knapsack problem, that is, the knapsack problem with a fractional objective function. After examining properties of optimal solutions of this problem, a new algorithm is given. This algorithm makes use of the greedy solution of the knapsack problem, which is generally only an approximate solution, but can be easily obtained. This algorithm is then compared with the straightforward modification of Dinkelbach's algorithm [Dinkelbach,'67]. It is theoretically shown that the former algorithm is always not worse than the latter, according to a certain measure. An upper bound on the number of required iterations is

also derived.

Chapter 7 provides two algorithms for quadratic fractional programming problems, both of which are based on Dinkelbach's approach. One is an algorithm using the Newton method that is often used to find zeros of a nonlinear function and the other is a straightforward search of the target points. These algorithms are implemented on computers, and the above algorithms are compared from the computational point of view.

Finally, Chapter 8 summarizes results obtained in this dissertation and discusses further directions of developments in discrete programming.

CHAPTER 2 AN ASSIGNMENT PROBLEM ON A NETWORK

2.1 Introduction

This chapter discusses a variant of the assignment problem. The ordinary assignment problem is to assign n persons to n jobs so that the sum of assignment costs should be minimized. Let c_{ij} be the assignment cost of assigning i-th person to j-th job for i=1,...,n, j=1,...,n. Mathematically, the problem can be formulated as follows:

$$\text{Minimize} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \qquad (2.1)$$

$$\text{subject to} \sum_{i=1}^{n} x_{ij} = 1, \quad j=1,\ldots,n, \qquad (2.2)$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i=1,\ldots,n, \qquad (2.3)$$

$$x_{ij} = 0 \text{ or } 1 \quad \begin{pmatrix} i=1,\ldots,n \\ j=1,\ldots,n \end{pmatrix} \qquad (2.4)$$

where x_{ij}'s are variables representing an assignment; $x_{ij} = 1$ when i-th person is assigned to j-th job and $x_{ij} = 0$ otherwise. Constraints (2.2) and (2.3) together mean that one and only one person is assigned to each job.

Assignment problem has a nice property that an optimal solution of its associated linear programming problem, where (2.4) is replaced with $x_{ij} \geq 0$, is also an optimal solution of the original problem. Based on this

property, H.W.Kuhn [Kuhn,'55] provided an efficient algorithm so called "Hungarian Method". Munkres also proposed a refinement of Kuhn's algorithm for the assignment and transportation problem.

Many variants of the above assignment problem are found in the literature [Gross,O.,'58] [Gilmore,'62] [Lawler,'63] [Pierskalla,W.P.,'68] [Charnes,A., et.al., '69].

We consider in this paper a variant which is different from those discussed in the above literature. This combines a network with the assignment problem. Consider the assignment of n persons (machines) to n jobs, the execution of which must satisfy certain precedence relations represented as a network. The objective to be minimized is the time of completion of all jobs. The problem is formally described as follows.

Let $N=[V,A]$ be an acyclic network with a set of vertices $V=\{v_0,v_1,v_2,\ldots,v_n,v_{n+1}\}$ and a set of arcs $A=\{a_1, a_2,\ldots,a_{\ell_0}\}\subset V\times V$. Each vertex represents a job and each arc a precedence relation between two jobs. That is, arc $a_\ell=(v_i,v_j)$ implies that execution of job v_j cannot be initiated prior to the completion of job v_i. It is noted that v_0 is the dummy job representing the start; hence no arc is incident to v_0. v_{n+1} is the dummy job representing the completion of all jobs; hence no arc is incident from v_{n+1}.

Let $M=\{m_1,m_2,\ldots,m_n\}$ be the set of n elements (persons or machines) to be assigned to the jobs. Suppose m_k is assigned to one of vertices $v_{\alpha(\ell)}$ in V to which

some vertex v_j is adjacent via a_ℓ, i.e., $a_\ell=(v_{\alpha(\ell)},v_j)$. Then $d^\ell_{k\alpha(\ell)}$ denotes the length of a_ℓ. That is, the length $d^\ell_{k\alpha(\ell)}$ of $a_\ell=(v_{\alpha(\ell)},v_j)$ is given by the execution time of job $v_{\alpha(\ell)}$ plus the setup time of v_j which generally depends on both $v_{\alpha(\ell)}$ and v_j. For simplicity, d^ℓ_{k0} (length of arc from v_0) is assumed to be zero.

The problem P is now defined as follows: Find an assignment of n elements in M to n jobs in V (i.e., one to one mapping: $M \rightarrow V$) which minimizes the length of the critical path (longest path) from v_0 to v_{n+1}. As a special case, if $N=[V,A]$ is given by $V=\{v_0,v_1,\ldots,v_{n+1}\}$ and $A=\{(v_0,v_1),(v_1,v_2),\ldots,(v_n,v_{n+1})\}$, the problem is reduced to the ordinary assignment problem.

The above problem naturally arises in optimal scheduling of a set of n jobs, on which a technological ordering described by a PERT-like network N is imposed. It is assumed that any person (or machine) can execute any job. For a given network N, the problem is to find an assignment of n persons with different ability, experiance, education and so forth (or n machines of different performance) in such a way that minimizes the total completion time of all n jobs. Note that the difference in persons (or machines) is represented solely by the lengths $d^\ell_{k\alpha(\ell)}$ of arc a_ℓ.

As an example, consider that network N represents the process of painting a house (or a toy, an automobile, etc) in n colours. Each vertex represents painting one colour. The precedence relation specified by N shows that the order in which n colours are painted; e.g.,

black is painted after white, etc. The length $d^{\ell}_{k\alpha(\ell)}$ of $a_{\ell}=(v_{\alpha(\ell)},v_j)$ is the sum of the time of painting colour $v_{\alpha(\ell)}$ and the time of waiting until it dries enough to start painting the next colour v_j. Then this problem is to assign n spray guns of varied performance to n vertices so that the entire process may be completed in the minimum time. (It is natural to assume that one spray gun is prepared for each colour.)

In the sequal, Section 2.2 gives a formulation of the problem as an integer programming problem. Section 2.3 describes the outline of the algorithm based on branch-and-bound method and the decomposition principle of Dantzig-Wolfe's type for the associated linear programming problem. Section 2.4 gives details of the method obtaining lower bound, and Section 2.5 discusses branching rule. Section 2.6 suggests further computational considerations. An illustrative example is also given in Section 2.7. Section 2.8 contains the result of computational experiances. Finally Section 2.9 summarizes Chapter 2.

2.2 Formulation as an Integer Programming Problem

The above assignment problem P is formulated as an integer programming problem in this section. An algorithm based on the branch-and-bound principle will then be developed in the subsequent sections. Branch-and-bound algorithms have attained certain success for other variants of the assignment problem such as the quadratic assignment problem [Gilmore,'62] [Lawler,'63] and the multidimensional assignment problem [Pierskalla,'68].

Let the path-arc matrix of N be given by $\mu=(\mu_{j\ell})$, where

$$\mu_{j\ell}=\begin{cases}1 & \text{if } a_\ell \in A \text{ is on the j-th path of } N\\ 0 & \text{otherwise}\end{cases}$$

$j=1,2,\ldots,j_0$ (= the number of paths in N)

$\ell=1,2,\ldots,\ell_0$ (= the number of arcs).

Introduce 0-1 variables x_{ki} which specify an assignment in the following manner:

$$x_{ki}=\begin{cases}1 & \text{if } m_k \in M \text{ is assigned to } v_i \in V\\ 0 & \text{otherwise.}\end{cases}$$

The length of the j-th path is then written as follows, for the assignment given by $x=(x_{11},x_{12},\ldots,x_{nn})$.

$$\sum_{\ell=1}^{\ell_0} \mu_{j\ell} \sum_{k=1}^{n} d^{\ell}_{k\alpha(\ell)}\, x_{k\alpha(\ell)}, \qquad j=1,2,\ldots,j_0.$$

(For notational convenience, dummy variables $x_{k0} = 0$ are introduced.) Thus P is formulated as the following LP problem with n^2 0-1 variables x_{ki} and one real variable λ.

P: minimize λ

$$\text{subject to} \sum_{\ell=1}^{\ell_0} \mu_{j\ell} \sum_{k=1}^{n} d_{k\alpha(\ell)}^{\ell} x_{k\alpha(\ell)} \leqq \lambda, \quad j=1,\ldots,j_0,$$

$$\sum_{i=1}^{n} x_{ki} = 1, \quad k=1,\ldots,n,$$

$$\sum_{k=1}^{n} x_{ki} = 1, \quad i=1,\ldots,n,$$

$$x_{ki} = 0 \text{ or } 1, \quad k,i=1,\ldots,n.$$

Note that P is the ordinary assignment problem if the first j_0 constraints having λ on their right hand sides are removed and the objective function is appropriately modified.

2.3 Solving Problem P by a Branch-and-Bound Method

Since the IP problem P is of considerably large size, we attempt to solve it by using a branch-and-bound method, rather than directly applying the existing integer programming algorithms. A branch-and-bound method (e.g., [Lawler and Wood,'66] [Mitten,'70]) is usually determined by the following two ingredients:

(a) Branching rule ; that specifies how to decompose a given (partial) problem P_j (P_j is either the original P or one of problems generated by this branching operation) into smaller partial problems $P_{j_1}, P_{j_2}, \ldots, P_{j_k}$ such that P_j can be solved if all $P_{j_1}, P_{j_2}, \ldots, P_{j_k}$ are solved;

(b) Bounding strategy ; that specifies how to attempt to solve P_j and, in case P_j is not solved by this attempt, how to obtain a lower bound of its optimal objective value.

Now we give a description of (a) and (b) for our problem, and after that a description of the whole branch-and-bound method will follow.

First the branching is made from a given (partial) problem P_j to two partial problems P_{j_1} and P_{j_2} such that

(i) P_{j_1} is obtained from P_j by adding the constraint that a certain $m_{\bar{k}} \in M$ is assigned to $v_{\bar{i}} \in V$, where no fixed assignment was given to $m_{\bar{k}}$ or $v_{\bar{i}}$ in P_j, and the assignment $m_{\bar{k}} \rightarrow v_{\bar{i}}$ was not prohibited in P_j ;

(ii) P_{j_2} is obtained from P_j by adding the constraint that the assignment $m_{\bar{k}} \rightarrow v_{\bar{i}}$ is prohibited.

This is formally described as follows. With a (partial) problem P_j, let two sets $F_j \subset M \times V$ and $H_j \subset M \times V$

be associated ; P_j is denoted by $P_j=(F_j,H_j)$. $(m_k,v_i) \in F_j$ denotes that the assignment $m_k \to v_i$ is fixed in P_j, while $(m_k,v_i) \in H_j$ denotes that the assignment $m_k \to v_i$ is prohibited in P_j. The original Problem P is defined by $P=(\phi,\phi)$. The branching is then defined by a pair $(m_{\bar{k}}, v_{\bar{i}})$ such that

$$(m_{\bar{k}},v_i) \notin F_j, \qquad i=1,2,\ldots,n$$

$$(m_k,v_{\bar{i}}) \notin F_j, \qquad k=1,2,\ldots,n$$

$$(m_{\bar{k}},v_{\bar{i}}) \notin H_j,$$

The resulting P_{j_1} and P_{j_2} are given by

$$P_{j_1}=(F_{j_1},H_{j_1})$$

$$P_{j_2}=(F_{j_2},H_{j_2}),$$

where

$$F_{j_1}=F_j \cup \{(m_{\bar{k}},v_{\bar{i}})\}$$

$$H_{j_1}=H_j$$

$$F_{j_2}=F_j$$

$$H_{j_2} = H_j \cup \{(m_{\bar{k}}, v_{\bar{i}})\}.$$

Since the way of selecting $(m_{\bar{k}}, v_{\bar{i}})$ for branching is crucial for the algorithm efficiency, it will be further discussed in Section 2.5.

To determine a bounding strategy, next, note that a partial problem P_j is also an IP problem obtained from P by fixing some variables to 0 or 1. Let $\bar{P}_j$ be the LP (linear programming) problem obtained from P_j by removing the integrality constraint. Obviously an optimal solution of $\bar{P}_j$ is an optimal solution of P_j if it is an integer solution of P_j. Furthermore, denoting optimal values of P_j and $\bar{P}_j$ by $\lambda(P_j)$ and $\lambda(\bar{P}_j)$ respectively, the following relation is easily proved:

$$\lambda(\bar{P}_j) \leq \lambda(P_j) .$$

Thus $\lambda(\bar{P}_j)$ is used as a lower bound of $\lambda(P_j)$.

Although $\bar{P}_j$ is usually a large LP problem, a decomposition technique can be applied since $\bar{P}_j$ is highly structured. The resulting computational procedure will be described in the next section.

A branch-and-bound algorithm for solving P is now given. In the algorithm, Problem P_j is called active if a conclusion that P_j can be discarded from the subsequent computation has been drawn for some reason, or a necessary step was already taken for P_j. The algorithm

terminates whenever no active problem exists.

Branch-and-Bound Algorithm for solving P (Algorithm A)

Step 1 (Initialization) : Let $P_0(=P)$ be active. Solve P_0. Set $\lambda^* \leftarrow \infty$. Solve $\bar{P}_0$. If $\bar{P}_0$ has an integer optimal solution, go to Step 4 after letting $\lambda^* \leftarrow \lambda(\bar{P}_0)$; otherwise go to Step 2.

Step 2 : Select one problem P_j, and go to Step 3. If no active problem exists, go to Step 4.

Step 3 (Branching and Bounding) : Decompose P_j into P_{j_1} and P_{j_2} by the branching rule described above, and terminate P_j. Solve $\bar{P}_{j_1}$ and $\bar{P}_{j_2}$. If $\bar{P}_{j_1}$ ($\bar{P}_{j_2}$) has an integer optimal solution, terminate P_{j_1} (P_{j_2}) and replace λ^* by $\lambda(\bar{P}_{j_1})$ ($\lambda(\bar{P}_{j_2})$) if $\lambda(\bar{P}_{j_1}) < \lambda^*$ ($\lambda(\bar{P}_{j_2}) < \lambda^*$). Terminate all active problem P_k (including P_{j_1} and P_{j_2}) such that

$$\lambda(\bar{P}_k) \geq \lambda^*$$

holds. Let P_{j_1} and/ or P_{j_2} not terminated in this process be active. Return to Step 2.

Step 4 : Terminate the computation. λ^* gives the value of optimal assignments.

The selection rule of an active problem P_j in Step 2 is also important in determinimg the algorithm efficiency. Although any rule discussed in the framework of general branch-and-bound methods would be applicable, so-called linear search rule that select the active problem which was most recently generated is employed in our

computation of Section 2.8. The linear search rule consumes less amount of memory space compared with others.

2.4 Solving Problem $\bar{P}_j$ for Obtaining Lower Bounds

For simplicity, an algorithm for solving $\bar{P}$ instead of $\bar{P}_j$ will be described. The modification necessary for general $\bar{P}_j$ is obvious. The essence of the following procedure is to apply the Dantzig-Wolfe decomposition technique [Dantzig,G.B. and P.C.Wolfe,'60] to $\bar{P}$ and reduce it to a series of small size LP problems (compared with $\bar{P}$) with two types of auxiliary problems one of which is the ordinary assignment problem and the other is the critical path problem. It is well known that considerably efficient algorithms exist for both the assignment problem [Kuhn,55] [Munkres,'57] and the critical path problem.

Let $x=(x_{11},x_{12},\ldots,x_{nn})$ and let

$$S=\{\, x \mid \sum_{i=1}^{n} x_{ki} = 1,\ \sum_{k=1}^{n} x_{ki} = 1,\ x_{ki} \geqq 0\}.$$

Denote extreme points of S by

$$x^r\ ,\quad r=1,2,\ldots,r_0\ (=n!),$$

i.e., x^r is an n^2 dimensional 0-1 vectors corresponding to assignments : $M \to V$. Any $x \in S$ can be represented as follows :

$$x = \sum_{r=1}^{r_0} \rho_r\, x^r$$

where

$$\sum_{r=1}^{r_0} \rho_r = 1,\ \rho_r \geqq 0,\quad r=1,2,\ldots,r_0.$$

By substituting these expressions, $\bar{P}$ is transformed into

$\bar{P}$: Minimize λ

$$\text{subject to} \sum_{r=1}^{r_0} \psi_{jr}\, \rho_r \leqq \lambda, \quad j=1,2,\ldots,j_0,$$

$$\sum_{r=1}^{r_0} \rho_r = 1,$$

$$\rho_r \geqq 0 \ , \quad r=1,2,\ldots,r_0.$$

where ψ_{jr} is a constant defined by

$$\psi_{jr} = \sum_{\ell=1}^{\ell_0} \mu_{j\ell} \sum_{k=1}^{n} d^{\ell}_{k\alpha(\ell)}\, x^{r}_{k\alpha(\ell)} \ ,$$

which is the length of the j-th path in N when assignment x^r is made.

$\bar{P}$ is usually a very large LP problem. However the following column and row generation technique makes it possible to solve $\bar{P}$ by a series of rather small LP problems. We will first show how the dual and primal feasibility can be checked without maintaining the entire simplex tableau of $\bar{P}$, and then give an algorithm for $\bar{P}$.

A Dual feasibility : For $J \subset J_0$ ($= \{1,2,\ldots,j_0\}$), let $\bar{P}(J)$ (the role of J will become clear later) be $\bar{P}$ with constraints

$$\sum_{r=1}^{r_0} \psi_{jr}\, \rho_r \leqq \lambda \qquad j \notin J$$

deleted. The modified costs $\bar{c}_r$ of $\bar{P}(J)$ are then given by

$$\bar{c}_r = \sum_{j \in J} \sigma_j \psi_{jr} - \sigma_0$$

where σ_j and σ_0 are the current values of dual variables (simplex multipliers) corresponding to

$$\sum_{r=1}^{r_0} \psi_{jr} \rho_r \leqq \lambda \qquad \sum_{r=1}^{r_0} \rho_r = 1,$$

respectively. Note that σ_j corresponding to

$$\sum_{r=1}^{r_0} \psi_{jr} \rho_r < \lambda$$

for the current values of ρ_r always assumes the value 0 due to the complementary slackness relation. Thus it is possible to assume that $j \in J$ always represents a path satisfying

$$\sum_{r=1}^{r_0} \psi_{jr} \rho_r = \lambda$$

since the addition of $\sigma_j = 0$ does not change $\bar{c}_r$.

The dual feasibility of $\bar{P}(J)$, i.e.,

$$\bar{c}_r \geqq 0, \quad r=1,\ldots,r_0$$

is satisfied if and only if

$$\bar{z} = min\{ \sum_{j \in J} \sigma_j \psi_{jr} \mid r=1,\ldots,r_0 \} \geqq \sigma_0$$

holds. Note that $\bar{z}$ is the optimal objective value of the following assignment problem :

$$\mathrm{D}(\sigma): \quad \text{Minimize} \quad z = \sum_{j \in J} \sigma_j \sum_{\ell=1}^{\ell_0} \mu_{j\ell} \sum_{k=1}^{n} d^{\ell}_{k\alpha(\ell)} x_{k\alpha(\ell)}$$

$$\text{subject to} \quad \sum_{k=1}^{n} x_{ki} = 1, \quad i=1,2,\ldots,n,$$

$$\sum_{i=1}^{n} x_{ki} = 1, \quad k=1,2,\ldots,n$$

$$x_{ki} \geq 0, \quad k=1,2,\ldots,n \quad i=1,2,\ldots,n,$$

where σ_j, $\mu_{j\ell}$, $d^{\ell}_{k\alpha(\ell)}$ are constants and x_{ki} are variables. Thus the dual feasibility of $\bar{\mathrm{P}}(J)$ can be checked by solving an ordinary assignment problem.

B **Primal feasibility** : For the current values of ρ_r, let the index set $R \subset R_0$ ($= \{1,2,\ldots,r_0\}$) be given such that $\rho_r = 0$ if $r \notin R$ (the role of R will become clear later). Assume that

$$\sum_{r \in R} \rho_r = 1 \text{ and } \rho_r \geq 0, \quad r \in R$$

hold. Define the weighted arc length of $a_\ell \in A$ for ρ by

$$d^{\ell}(\rho) = \sum_{r \in R} \rho_r \sum_{k=1}^{n} d^{\ell}_{k\alpha(\ell)} x^{r}_{k\alpha(\ell)} \quad \ell=1,2,\ldots,\ell_0.$$

Let $N(\rho)$ be the network N with each $a_\ell \in A$ having length $d^{\ell}(\rho)$. Then the primal feasibility of $\bar{\mathrm{P}}$ is satisfied if

and only if all path lengths in $N(\rho)$ are not greater than λ (=the current objective value). The last condition is satisfied if and only if the length $v(\rho)$ of critical paths in $N(\rho)$ satisfies

$$v(\rho) \leqq \lambda.$$

Thus the primal feasibility of $\bar{P}$ can be checked by solving a critical path problem.

Before proceeding to an algorithm for solving $\bar{P}$, define the LP problem $\bar{P}(J,R)$ for $J \subset J_0$ and $R \subset R_0$.

$\bar{P}(S,R)$: minimize λ

$$\text{subject to} \quad \sum_{r \in R} \psi_{jr} \rho_r \leqq \lambda, \quad j \in J$$

$$\sum_{r \in R} \rho_r = 1,$$

$$\rho_r \geqq 0, \quad r \in R,$$

Algorithm for solving Problem $\bar{P}$ (Algorithm B)

Step 1 : Start with $J \subset J_0$ and $R \subset R_0$ heuristically obtained. Go to Step 2.

Step 2 : Solve $\bar{P}(J,R)$ and denote its optimal tableau by T. Redefine R by $R:=R-\tilde{R}$ (i.e., delete from T all columns corresponding to $r \in \tilde{R}$), where $\tilde{R}$ is the set of indices r such that ρ_r is nonbasic and $\bar{c}_r > 0$ in T. Let the resulting tableau be T and go to Step 3.

Step 3 : If T is dual feasible (this can be checked by solving the assignment problem $D(\sigma)$), go to Step 4. Otherwise let $R \leftarrow R \cup \{\tilde{r}\}$ (i.e., augment T by column $\tilde{r}$), where $x^{\tilde{r}}$ is the optimal assignment of $D(\sigma)$, and return to Step 2.

Step 4 : If T is primal feasible (this can be checked by obtaining a critical path of $N(\rho)$), terminate. T gives an optimal solution of $\bar{P}$. Otherwise, let $J \leftarrow J \cup \{\tilde{j}\} - \tilde{J}$ (i.e., added row $\tilde{j}$ is the index of the critical path of $N(\rho)$, and $\tilde{J}$ is defined as follows : $\tilde{J}$ is the set of indices corresponding to non-binding constraints of T (i.e., with positive slack variables) if the objective value λ was improved after the previous dual feasible solution ; $\tilde{J} = \phi$ if the objective value λ was not improved or the current solution is the first dual feasible solution. Return to Step 2.

The finite convergence of this algorithm under the usual nondegeneracy assumption can be proved in a similar way to the one used by [Geoffrion,'70] for proving the convergence of the general relaxation strategy.

A good heuristic rule for defining the initial J and R in Step 1 may be to let J be the set of indices for critical paths of N with all arc lengths considered to be 1, and to let R be a set of indices for assignments which make these paths in J reasonably short. In the experiment of Section 2.8, however, a much simpler rule was employed.

2.5 Branching Rule

Various methods would be conceivable for determining the pair $(m_{\bar{k}}, v_{\bar{i}})$ which defines the branching operation in Step 3 of the branch-and-bound method outlined in Section 2.3. The following used in our computation should be one of the most reasonable ones.

Let LP problem $\bar{P}_j$ have the optimal solution :

$$\bar{\rho} = (\bar{\rho}_1, \bar{\rho}_2, \ldots, \bar{\rho}_{r_0}).$$

Based on $\bar{\rho}$, define

$$x_{ki}(\bar{\rho}) = \sum_{r=1}^{r_0} \bar{\rho}_r \, x_{ki}^r$$

$$x_{\bar{k}\bar{i}}(\bar{\rho}) = \max_{k,i} \{ x_{ki}(\bar{\rho}) \mid 0 \leq x_{ki}(\bar{\rho}) < 1 \}.$$

$x_{\bar{k}\bar{i}}(\bar{\rho})$ is considered to indicate the most promising assignment $m_{\bar{k}} \rightarrow v_{\bar{i}}$, as far as the information contained in $\bar{\rho}$ is concerned. Thus $(\bar{k}, \bar{i})$ is used to define the branching from P_j. The case of $x_{ki}(\bar{\rho}) = 1$ is excluded since it is likely that the partial problems generated from P_j also satisfy $x_{ki} = 1$ even without being fixed. (If x_{ki} was already fixed to 1 by the branching operation applied to P_k, from which P_j resulted, we obviously need not fix it again. The above rule also excludes this possibility.)

2.6 Further Computational Considerations

In implementing the algorithm outlined so far, the following remarks may be useful from the view point of computational speed.

(i) The value of each dual feasible solution of $\bar{P}_j$ obtained in Step 3 of Algorithm B (Section 2.4) is not greater than the optimal value $\lambda(\bar{P}_j)$. Thus the value of any dual feasible solution can be used as a lower bound of $\lambda(\bar{P}_j)$. This suggests the termination of the LP computation (Algorithm B) before the primal feasibility is attained. A reasonable cutoff rule may be to terminate the computation of $\bar{P}_j$ whenever a dual feasible tableau T with $\upsilon(\rho) - \lambda \leqq \varepsilon$ is obtained, where λ is the objective value of T, $\upsilon(\rho)$ is the length of a critical path of $N(\rho)$ and $\varepsilon (\geqq 0)$ is a given constant (note $\upsilon(\rho) - \lambda \leqq 0$ implies primal feasibility). $\varepsilon = 10$ was used in our experiment of Section 2.8. (See also the discussion given to Table 2.3 and Table 2.4 of Section 2.8.)

(ii) In solving $\bar{P}_j$ by Algorithm B, or in passing from problem $\bar{P}_j$ to $\bar{P}_k$, where $\bar{P}_k$ is obtained from $\bar{P}_j$ by branching or backtracking, the technique of parametric programming (or sensitivity analysis) known in the LP theory can be fully utilized. Details are, however, omitted.

(iii) The optimal solution (or a good feasible solution) of $\bar{P}_j$ may be used to generate good assignments ; assignment x^r with relatively large $\bar{\rho}_r$ in $\bar{\rho}$ would have a high probability of being close to an optimal assignment of P. Thus in Step 3 of the branch-and-bound algorithm (Algorithm A) in Section 2.3, a certain number of assign-

ments considered promising according to optimal LP solutions of $\bar{P}_{\cdot j_1}$ and $\bar{P}_{\cdot j_2}$ may be actually constructed and tested. If an assignment with a smaller objective value than the current λ^* is found, λ^* is immediately replaced by the new value. This modified algorithm has a tendency of keeping the value of λ^* smaller than that without this modification. Hence a speed up in computation time can be expected. In our experiment of Section 2.8, all new assignments x^r with positive ρ_r were actually tested.

2.7 Example

Consider the problem with the network of Figure 2.1 and arc lengths $d^{\ell}_{k\alpha(\ell)}$ given by Table 2.1.

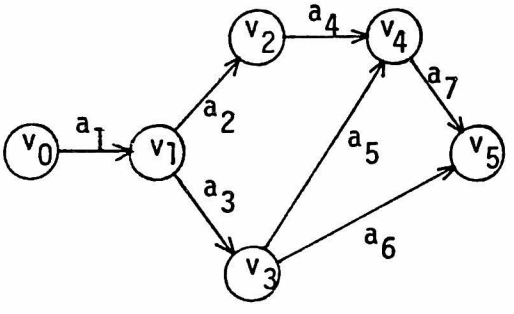
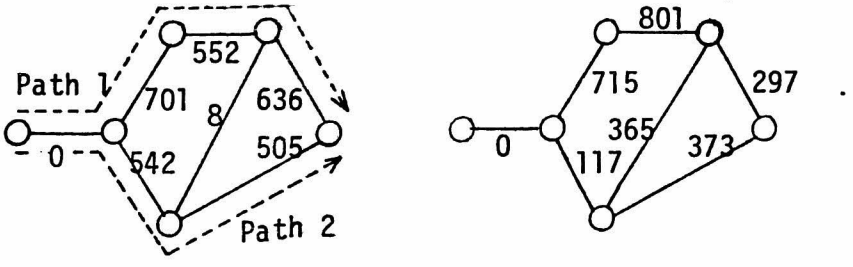


Figure 2.1. Precedence relations between jobs.

Table 2.1. Arc lengths $d^{\ell}_{k\alpha(\ell)}$ of the assignment problem on network of Figure 2.1.

ℓ	2	3	4	5	6	7
$\alpha(\ell)$	1	1	2	3	3	4
k=1	701	542	801	422	0	404
2	408	308	552	365	373	225
3	224	947	874	8	505	297
4	715	117	512	186	324	636

After letting $\lambda^* = \infty$ in Step A-1 (i.e., Step 1 of Algorithm A), Step B-1 (i.e., Step 1 of Algorithm B) is entered to solve $\bar{P}_0$. The initial J of Step B-1 is given by path 1 and path 2 of Figure 2.2, and the initial R by assignment 1 and assignment 2 of Figure 2.2, where $(i_1\ i_2\ i_3\ i_4)$ denotes the assignment $\{ m_1 \to v_{i_1}, m_2 \to v_{i_2}, m_3 \to v_{i_3}, m_4 \to v_{i_4} \}$. The arc lengths for these assignments are also shown in Figure 2.2.



Assignment 1 =(1234)

ψ_{11} =1889

ψ_{12} =1047

Assignment 2 =(2341)

ψ_{12} =1813

ψ_{22} = 490

Figure 2.2. Initial paths and assignments

This defines LP problem $\bar{P}(J,R)$ as follows :

$\bar{P}(\{1,2\},\{1,2\})$: Minimize λ

subject to $1889\ \rho_1 + 1813\ \rho_2 \leqq \lambda$,

$1049\ \rho_1 + 490\ \rho_2 \leqq \lambda$,

$$\rho_1 + \rho_2 = 1 ,$$

$$\rho_1 , \rho_2 \geqq 0 .$$

An optimal solution (obtained by the simplex method) is

$$\lambda=1813,\ \rho_1=0,\ \rho_2=1,\ \sigma_0(=\lambda) = 1813,\ \sigma_1 = 1,\ \sigma_2 = 0,$$

where σ_i are dual variables. The non-basic column ρ_1 is then deleted since $\bar{c}_1 > 0$ as easily calculated (Step B-2).

To check the dual feasibility of this solution (Step B-3), assignment problem $D(\sigma)$ with $\sigma_1 = 1$ is then solved. Coefficients of $D(\sigma)$ are shown in Table 2.2. The (k,i)-th element of Table 2.2 shows the coefficient of x_{ki} in the objective function of $D(\sigma)$.

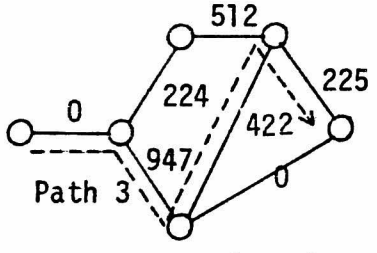
Table 2.2. Coefficients of the assignment problem to check the dual feasibility.

k \ i	1	2	3	4
1	701	801	0*	404
2	408	552	0	225*
3	224*	874	0	297
4	715	512*	0	636

An optimal assignment of $D(\sigma)$ (indicated by * in Table 2.2) is

assignment 3 = (*3412*)

and the optimal value is *961*. Arc lengths for assignment 3 are shown in Figure 2.3.



Assignment 3 =(3412)

ψ_{13} =961

ψ_{23} =947

Figure 2.3. Arc lengths for assignment 3.

Since $\bar{z} = 961 < 1813 = \sigma_0$ holds, the present solution is not dual feasible. Thus assignment 3 is added to form the new $\bar{P}(J,R)$, that is

$\bar{P}(\{1,2\},\{2,3\})$: Minimize λ

subject to $1813\ \rho_2 + 961\ \rho_3 \leq \lambda$,

$591\ \rho_2 + 947\ \rho_3 \leq \lambda$,

$$\rho_2 + \rho_3 = 1,$$

$$\rho_2 , \rho_3 \geq 0.$$

An optimal solution obtained in Step B-2 is

$$\lambda = 961,\ \rho_2=1,\ \rho_3=1,\ \sigma_0 = 961,\ \sigma_1 = 1,\ \sigma_2 = 0.$$

(Although $\bar{P}(\{1,2\},\{2,3\})$ is actually solved starting with the optimal tableau of $\bar{P}(\{1,2\},\{1,2\})$ to facilitate the computation, the details are omitted for simplicity.) The non-basic column ρ_2 is then deleted since $\bar{c}_r > 0$ (Step B-2).

The assignment problem $D(\sigma)$ to check the primal feasibility is the same as the above one with $\bar{z} = 961$. Thus $\bar{z} = \sigma_0$ holds and the present solution is dual feasible (Step B-3).

Now it is required to check the primal feasibility (Step B-4). Since $\rho_3=1$ implies that network $N(\rho)$ is the same as the one in Figure 2.3 corresponding to assignment 3, this is done by calculating its critical path (path 3). Path 3 has the length $v(\rho) = 1594$. $v(\rho) = 1594 > 961 = \lambda$ shows that the present solution is not primal feasible.

We again return to Step B-2 and solve $\bar{P}(\{1,2,3\},\{3\})$ augmented by path 3. Repeating Step B-2 and Step B-3, the next dual feasible solution is obtained after adding two more assignments :

assignment 4 = (2431), and assignment 5 = (4132).

In this case, the resulting solution is also primal feasible. Thus its optimal solution

$$\lambda(=\lambda(\bar{P}_0))=1146.7,\ \rho_3=0.49,\ \rho_5=0.51$$

also solves $\bar{P}_0$. Figure 2.4 shows network $N(\rho)$ for this solution, in which path 1 and path 3 are both critical.

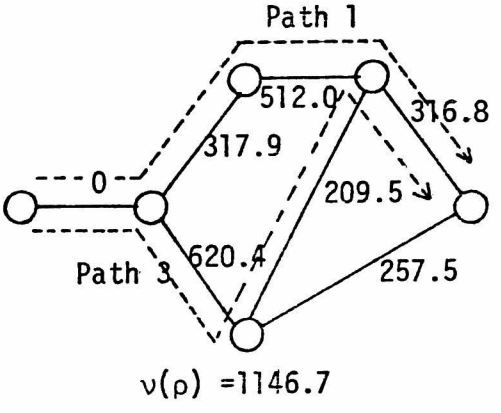


Figure 2.4. Network $N(\rho)$ corresponding to the optimal solution of $\bar{P}_0$.

At this point, assignment 3 and assignment 5 (with positive ρ_r) are tested to improve λ^* (see remark (iii) of Section 2.6). Assignment 5 gives a smaller value than assignment 3. Its value 1324 is hence stored as the new λ^*. This completes the computation associated with the initial problem P_0.

Returning to Algorithm A, $\bar{P}_0$ is selected in Step A-2

since it is the only active problem. The branching of Step A-3 is done according to the rule in Section 2.5. $x_{ki}(\bar{\rho})$ for $\bar{\rho}_3 = 0.49$, $\bar{\rho}_5 = 0.51$ (optimal solution of $\bar{P}_0$) are first calculated. They are

$$x_{13}(\bar{\rho}) = x_{24}(\bar{\rho}) = x_{31}(\bar{\rho}) = 0.49,$$

$$x_{14}(\bar{\rho}) = x_{21}(\bar{\rho}) = x_{33}(\bar{\rho}) = 0.51,$$

$$x_{42}(\bar{\rho}) = 1, \; x_{ki}(\bar{\rho}) = 0 \text{ for other } k \text{ and } i.$$

Thus $x_{14}(\bar{\rho}) = 0.51$ which is the first one with the highest value (excluding $x_{42}(\bar{\rho}) = 1$) is selected to define the branching $m_1 \rightarrow v_4$ and $m_1 \not\rightarrow v_4$.

The subsequent branch-and-bound computation proceeds as illustrated in Figure 2.5. Node numbers indicate the order in which partial problems are tested. λ^* attached to each node is the value when the computation of that node is completed. $\lambda(\bar{P}_j)$ is the LP optimal value of P_j and it works as a lower bound. When node 6 is tested, no active node exists since all node have lower bounds greater than $\lambda = 1217$. Hence the whole computation terminates (Step A-4).

The optimal solution is

assignment 7 = (3142)

with $\lambda = 1217$, which was obtained at node 2 in the computation of the new λ^*. This optimal solution is shown in

Figure 2.6.

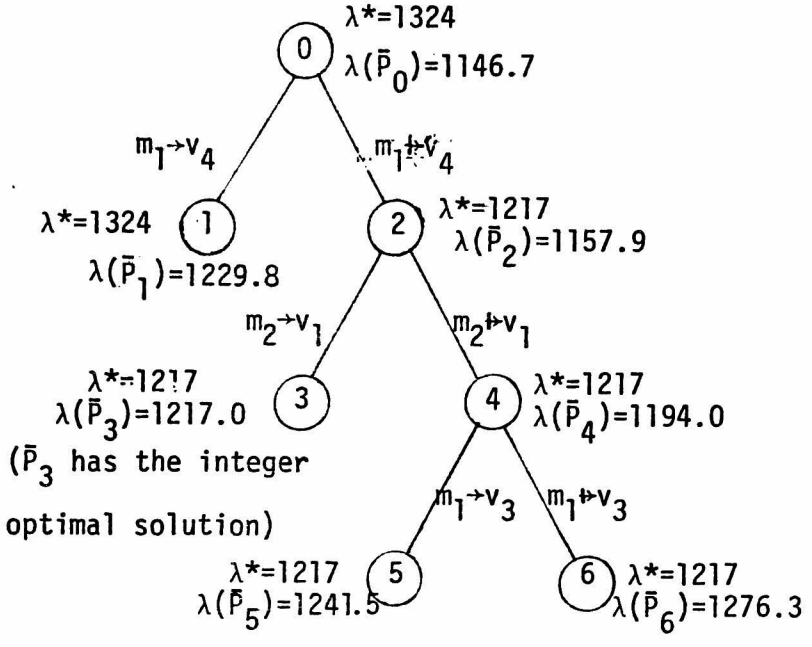
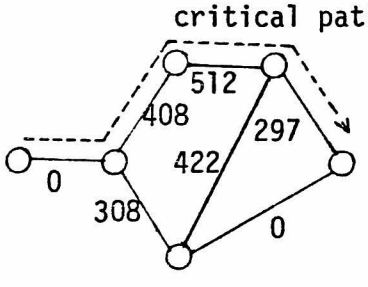


Figure 2.6. Illstration of the branch-and-bound computation.



Assignment 7 =(3142)

λ =1217

Figure 2.6. Optimal assignment.

2.8 Computational Experiment

To see the efficiency of our algorithm, the entire procedure was coded in *FORTRAN* and run *FACOM 230/75* computer at Kyoto University. The machine roughly corresponds to *IBM 370/165*. The code for ordinary assignment problems was obtained by translating the Silver's code [Silver,P.,'60] written in *ALGOL*.

Before determining the details of the branch-and-bound algorithm (Algorithm A), the possibility of strategy (i) of Section 2.6 was tested by solving LP problems $\bar{P}_j$ (using Algorithm B) and examining the quality of the values of dual feasible solutions which are not necessarily primal feasible. Results are shown in Table 2.3. Each figure is the average of 11 to 40 problems as indicated in the bottom row. All problems were generated randomly by assigning lengths taken from the uniform distribution between 0 and 1 to arcs of three networks : 10A, 10B and 20A of Table 2.3. The initial J and R of Step B-1 were defined as follows.

J represents two paths : One with the largest number of arcs, and the other with the largest number of arcs in the graph resulted by removing the first path.

R represents two assignments : $(1,2,\ldots,n)$ and $(2, n-1,4,n-3,6,n-5,\ldots,n,1)$.
(These are adopted mainly for the sake of simplicity.) It may be concluded that rather accurate lower bounds can be expected even if the computation is cut off prior to the primal feasibility (90% estimation attained in about a half of the total number of columns).

Table 2.3. Computational results for LP problems $\bar{P}_j$.

Problem type		10A	10B	20A
Number of vertices (persons) n		10	10	20
Number of arcs ℓ_0		13	18	26
Number of generated columns (assignments) (a)		13.0	37.2	63.0
Number of generated rows (paths) (a)		3.2	6.5	5.9
90 % point[(b)]	columns	10.1	17.0	35.7
	rows	2.7	3.5	4.4
Number of test problems		40	23	11

Table 2.4. Results for assignment problems P with ε=0 and ε=10.

Problem type	10B		10B	
ε	0	10	0	10
Computation time in seconds	2.43	2.25	4.84	4.39
Number of generated partial problems (c)	42	48	52	54

Table 2.5. Computational results for assignment problems on networks by branch-and-bound.

Problem type		10A	10B	20A
Number of generated partial problems (c)	Mean	73.9	140.2	891.6
	Standard deviation	107.9	152.5	1013.8
Number of generated columns (assignments) (a)		223	631	7256
Number of generated rows (paths) (a)		5	49	27
Number of pivots		390	1507	13491
When an optimal assignment generated (d)		126	104	436
Computation time in seconds		1.9	7.2	193.0
Number of test problems		30	20	10

Notes to tables

(a) Some columns (rows) may be counted repeatedly if deleted columns (rows) are again added back to set R (J).

(b) Numbers of columns and rows when a dual feasible solution with objective value $\lambda \geqq 0.9\lambda(\bar{P})$ is attained.

(c) This number contains those partial problems which are terminated in Step B-3.

(d) The number of columns generated before the column corresponding to an optimal assignment is generated.

Strategy (i) of Section 2,6 is further justified by Table 2.4, which shows the computational results for two type 10B problems (type 10A is too simple to see the difference) using both ε=0 and ε=10. (Arc lengths are defined in the same manner as in Table 2.5 described next.) It seems that setting ε=10 tends to consume less computation time although it generates a slightly larger number of partial problems.

Based on these preliminary results, ε in the cutoff was set to 10. Then a number of problems was solved by Algorithm A with LP subalgorithm (Algorithm B). All test problems were also generated randomly, but in this case integer lengths taken from the uniform distribution between 0 and 999 were assigned to arcs. Since each problem in this case has an integer optimal value, λ of each dual feasible solution can be used as a lower bound after being rounded up to the smallest integer not smaller than λ. (This gimmick was quite useful to increase the efficiency.)

Computational results are summarized in Table 2.5. Problem type in the first row refers to the same network as Table 2.3. Each figure in Table 2.5 shows the average of 10 $\sim$30 problems. It was noticed that the behavior of the algorithm was quite erratic. The standard deviations of the number of partial problems are also included to indicate this.

In most cases, relatively good assignments (if not optimal) were obtained in the early stage of the computation (see (iii) of Section 2.6 and the row labelled "when

an optimal assignment generated" in Table 2.5.) This indicates that our algorithm is also useful as a suboptimal algorithm in case enough computation time is not available.

Unfortunately it seems difficult to obtain exact optimal solutions of large problems (say, $n=30$), due to the rapid increase in the computation time. Probably this is because the quality of lower bounds provided by LP optimal solutions are not accurate enough. Thus it would be a direction of future research to find methods for obtaining better lower bounds.

2.9 Conclusion

A variant of the assignment problem which seeks an optimal assignment in a given network was proposed in this chapter. The problem was first formulated as an integer programming problem. Then the algorithm based on branch-and-bound method and the decomposition principle of Dantzig-Wolfe type for the associated linear programming problem was given. The algorithm makes use of lower bounds obtained from the generated linear programming problems. Each linear programming problem was solved by reducing it to a series of rather smaller linear programming problem with two types of auxiliary problems; the ordinary assignment problem and the critical path problem.

Though both the assignment problem and the critical path problem have efficient algorithms which run in a polynomial time of the problem size n, the entire algorithm for solving the linear programming problem is not very efficient according to the computational experiment in Section 2.8. Two reasons may be conceivable for this behavior; (i) the Dantzig-Wolf decomposition algorithm tends to slow down the speed of improvement in the objective value when it gets close to the optimal value, and (ii) our implementation (especially data structure) does not completely exploit the problem structure. This point should be further pursued in the future research.

As is stated in Section 2.1, many variants of the assignment problem have been considered. But almost all of them lost the property that LP relaxation solves the

original problem. Since this property is crucial for developing efficient algorithms for the assignment problem, some other variations without losing this property may be worth investigating.

CAHPTER 3 MINIMAX TYPE TRAVELING SALESMAN PROBLEMS

3.1 Introduction

This chapter treats a delivery route problem, in which a truck delivers some amount of production resource to each of n cities starting from the specified initial city. The time required to deliver the resources from one city to another is known. At each city, the production is started upon receiving the production resources and takes a certain time until completion.

The problem is to find a delivery route that visits each of the cities exactly once such that the completion time of production at all cities is minimized. This problem can be formulated in a form similar to the well-known traveling salesman problems.

The ordinary traveling salesman problem may be described as follows: A salesman travels n-cities exactly once and returns to the starting city. Given each travel time t_{ij} from city i to city j ($i \neq j$, i=1,2,$\cdots$,n j=1,2, $\cdots$,n), find the order of cities 1,2,$\cdots$,n minimizing the total traveling time.

For solution procedures of the above ordinary traveling salesman problem, algorithms based on the dynamic programming (Held and Karp [Held and Karp,'62], Bellman [Bellman,'62b] etc) and on branch-and-bound method (Little et.al.,[Little et.al.,'63] and Held and Karp [Held and Karp,'72 and '71]) are known. As variants of the traveling salesman problem, "bottleneck traveling salesman problem" by D. Shapiro [Shapiro, D.,'66] (its

purpose is to minimize the maximum travel time between two successively visited cities in a tour) and one with special cost given in an integral form by Gilmore and Gomory [Gilmore and Gomory], etc. have been considered. The traveling salesman problem discussed in this chapter, however, seems to be new because a minimax type objective function is used instead of the ordinary linear one. The known procedures such as cited above do not seem to be directly applied.

First in Section 3.2, our routing problem is formulated as a minimax type traveling salesman problem. After giving some notations and terminologies in Section 3.3, the problem is decomposed into (n-1) subproblems in Section 3.4. A solution procedure for each subproblem is also given in the same section. Finally, in Section 3.5, an entire algorithm is given to solve the minimax type traveling salesman problem. An illustrative example is given in Section 3.6. Section 3.7 summarizes the results in this chapter.

3.2 Formulation of a Minimax Type Traveling Salesman Problem

Let a truck deliver production resources to cities $i=1,2,\ldots,n$, starting from the city 1. Delivery from city i to city j takes t_{ij} unit times. At each city i, the production takes t_i unit times until the completion.

Rearranging the city numbers and $t_1 \geqq 0$,

$$t_2 \geqq t_3 \geqq \cdots \geqq t_n \geqq 0$$

can be assumed without loss of generality. Let Π_n denote the set of all permutations π on $V=\{1,2,\ldots,n\}$ and let $\tilde{\Pi}_n=\{\pi \in \Pi \mid \pi(1)=1\}$. Each $\pi \in \tilde{\Pi}_n$ corresponds to a delivery route, that is, $\pi(i)$ gives the i-th visited city. The production completion time at each city $\pi(k)$ $(k=2,\ldots,n)$, denoted by $f^{\pi}_{\pi(k)}$, becomes as follows:

$$f^{\pi}_{\pi(k)} \triangleq \sum_{i=1}^{k-1} t_{\pi(i)\pi(i+1)} + t_{\pi(k)} \quad .$$

Thus the completion time f^{π} at all cities becomes

$$f^{\pi} \triangleq max\{ f^{\pi}_{\pi(k)} \mid k=2,3,\ldots,n\} \quad .$$

Our problem is first stated as follows.

Problem A: Find $\pi \in \tilde{\Pi}_n$ minimizing f^{π} .

Setting

$$c_{ij} = t_{ij} + t_j - t_i \quad , \quad (1 \leqq i \leqq n,\ 1 \leqq j \leqq n,\ i \neq j),$$

we have

$$f^{\pi}_{\pi(k)} = \sum_{i=1}^{k-1} (t_{\pi(i)\pi(i+1)} + t_{\pi(i+1)} - t_{\pi(i)})$$

$$= \sum_{i=1}^{k-1} c_{\pi(i)\pi(i+1)} .$$

Now let $G=(V,E)$ be a complete directed graph with a vertex set $V=\{v_1,v_2,\ldots,v_n\}$, an arc set $E=\{(v_i,v_j) \mid v_i, v_j \in V, v_i \neq v_j\}$ and an arc cost c_{ij} associated with each arc (v_i,v_j) (hereafter v_i is denoted by i only and arc (v_i,v_j) by (i,j) unless misunderstanding is caused). Then $\pi \in \Pi^{\gamma}_{n}$ corresponds to a hamiltonian path $\pi=(\pi(1),\pi(2),\ldots,\pi(n))$ on G (hereafter π is used to denote both a permutation and the corresponding hamiltonian path) and $f^{\pi}_{\pi(k)}$ corresponds to the length f^{π}_{γ} of the route from vertex $\pi(1)=1$ to vertex $\gamma=\pi(k)$ on the given hamiltonian path π.

The above discussion shows that problem A is equivalent to the following problem B.

Problem B : Find a hamiltonian path $\pi \in \Pi^{\gamma}_{n}$ minimizing

$$max[\, f^{\pi}_{\gamma} \mid \gamma \in \{2,3,\ldots,n\}].$$

3.3 Notations and Definitions

Before proceeding to the algorithm, we introduce some notations to facilitate the subsequent discussion.

(1) $Q \subseteq V-\{1\}$: a subset of V.

(2) $|Q|$: the cardinary of set Q.

(3) $\Pi(\alpha,Q,\beta)$: a subset of $\hat{\Pi}_n$ defined by

$$\begin{cases} \text{(a) } \pi(i) = \alpha \quad \text{for } 1 \leq i \leq n-|Q|-1 \\ \text{(b) } \pi(i+j) \in Q, \quad j=1,2,\ldots,|Q| \\ \text{(c) } \pi(i+|Q|+1) = \beta \end{cases}$$

where

$$Q \subset V-\{1\}, \quad \alpha,\beta \in V-Q \ .$$

(4) $f^{\pi}(\alpha,Q,\beta)$: the length of the portion from vertex α to vertex β on path $\pi \in \Pi(\alpha,Q,\beta)$, i.e.,

$$f^{\pi}(\alpha,Q,\beta) = \sum_{k=i}^{i+|Q|} c_{\pi(k)\pi(k+1)}$$

where $\pi(i) = \alpha$.

(5) $\theta(\pi)$: the vertex with the smallest number among

ones giving

$$max\{ f^{\pi}_{\pi(k)} \mid k=1,2,\ldots,n\}.$$

(6) $\Pi(\delta)$: the subset of $\hat{\Pi}_n$ consisting of permutations $\pi \in \hat{\Pi}_n$ satisfying $\theta(\pi) = \delta$.

Note that $\theta(\pi)$ is the vertex δ maximizing f^{π}_{δ} on the hamiltonian path π and that the condition $\theta(\pi) = \delta$ is equivalent to the following conditions:

$$\begin{cases} \text{(a) } \pi(i)=\alpha \\ \text{(b) } f^{\pi}(\pi(j),\{\pi(j+1),\ldots,\pi(i-1)\},\delta) > 0, \quad \text{for } j=1,\ldots,i-1. \\ \text{(c) } f^{\pi}(\delta,\{\pi(i+1),\ldots,\pi(i+j-1)\},\pi(i+j)) \leqq 0 \quad \text{for } j=1,\ldots,n-i. \end{cases}$$

3.4 Decomposition of B into Subproblems $P_2, P_3, \ldots, P_n$

Problem B is decomposed into the following subproblems $P_2, P_3, \ldots, P_n$ based on the concept of $\Pi(\delta)$ defined in the previous section.

Problem P_δ: Find a hamiltonian path $\pi \in \Pi(\delta)$ minimizing f_δ^π, the length of the initial portion from vertex 1 to vertex δ on π.

Hereafter an optimal solution of Problem P_δ is denoted by π_δ. Note that $f^\pi = f_\delta^\pi$ holds for $\pi \in \Pi(\delta)$ from the concept of $\Pi(\delta)$.

Theorem 3.1 Among optimal solutions π_δ of each subproblems P_δ, $\delta=2,3,\ldots,n$, π^* giving

$$f^{\pi^*} = min\{ f^{\pi\delta} \mid \delta=2,3,\ldots,n\}$$

is also an optimal solution of Problem B.

Proof: By definition, we have

$$\overset{\sim}{\Pi}_n = \Pi(2) \cup \ldots \cup \Pi(n), \quad \Pi(i) \cap \Pi(j) = \phi \quad (i \neq j).$$

Therefore Problem B is equivalent to

$$\min_{\pi \in \overset{\sim}{\Pi}_n} \max_{\gamma \in \{1,2,\ldots,n\}} f_\gamma^\pi$$

$$= \min_{\pi\in\Pi(2)\cup\ldots\cup\Pi(n)} \max_{\gamma\in\{2,3,\ldots n\}} f_\gamma^\pi$$

$$= \min_{\delta\in\{2,3,\ldots,n\}} \min_{\pi\in\Pi(\delta)} \left(\max_{\gamma\in\{2,3,\ldots,n\}} f_\gamma^\pi\right)$$

$$= \min_{\delta\in\{2,3,\ldots,n\}} \left[\min_{\pi\in\Pi(\delta)} f_\delta^\pi\right] = \min_{\delta\in\{2,3,\ldots,n\}} f^{\pi\delta} \quad \square$$

In the following, some properties of P_δ useful for facilitating our algorithm are deduced.

Proposition 3.1. Let $\pi(i)=\delta$ for $\pi \in \Pi(\delta)$ and let

$$V(\delta) \triangleq \{\gamma \mid t_\gamma > t_\delta\} \cup \{1\}.$$

Then, for any vertex $\gamma \in V(\delta)$, there exists a certain $j < i$ satisfying $\pi(j) = \gamma$. (In other words, $\gamma \in V(\delta)$ appears before δ on π.)

Proof: Suppose that there exists a vertex $\gamma \in V(\delta)$ such that $\pi(j) = \gamma$ holds $j > i$. Then

$$f_\gamma^\pi - f_\delta^\pi = f_{\pi(j)}^\pi - f_{\pi(i)}^\pi = \sum_{k=i}^{j-1} c_{\pi(k)\pi(k+1)}$$

$$= \left(\sum_{k=i}^{j-1} t_{\pi(k)\pi(k+1)}\right) + t_\gamma - t_\delta > 0,$$

since $t_\gamma - t_\delta > 0$ ($\gamma \in V(\delta)$) and $\sum_{k=i}^{j-1} t_{\pi(k)\pi(k+1)} \geq 0$.

This implies that $\pi \in \Pi(\delta)$, and it contradicts the property

$\theta(\pi) = \delta$. □

Proposition 3.2. Consider $\pi \in \Pi(\alpha,Q,\delta) \cap \Pi(\delta)$ with $\pi(k) = \alpha$ (see Figure 3.1). If there exists $\hat{\pi} \in \Pi(\alpha,Q,\delta)$ satisfying

$$\begin{cases} \text{(a)} \quad \hat{\pi}(i) = \alpha \\ \text{(b)} \quad f^{\hat{\pi}}(\alpha,Q,\delta) \leqq 0 \\ \text{(c)} \quad f^{\hat{\pi}}(\hat{\pi}(i+j),\{\hat{\pi}(i+j+1),\ldots,\hat{\pi}(i+|Q|+1)\},\delta) \geqq 0 \\ \qquad\qquad\qquad\qquad \text{for all } j=1,2,\ldots,|Q|, \end{cases}$$

then there exists $\hat{\hat{\pi}} \in \Pi(\beta)$ satisfying

$$f^{\hat{\hat{\pi}}} \leqq f^{\pi}$$

for $\beta = \pi(\ell)$, $\ell \leqq k$.

Proof: The following $\hat{\hat{\pi}} \in \overset{\vee}{\Pi}_n$ satisfies the above conditions. (See Figure 3.2).

$$\hat{\hat{\pi}} : \begin{cases} \text{(d)} \quad \hat{\hat{\pi}}(j) = \pi(j),\ j=1,2,\ldots,k, \text{ and } k+|Q|+1,\ldots,n \\ \text{(e)} \quad \hat{\hat{\pi}}(k+j) = \hat{\pi}(i+j), \quad j=1,2,\ldots,|Q|. \end{cases}$$

By definition, the following properties are obvious.

$$f^{\hat{\hat{\pi}}}_{\hat{\hat{\pi}}(j)} = f^{\pi}_{\pi(j)}, \quad j=1,2,\ldots,k \quad . \qquad (3.1)$$

In particular we have

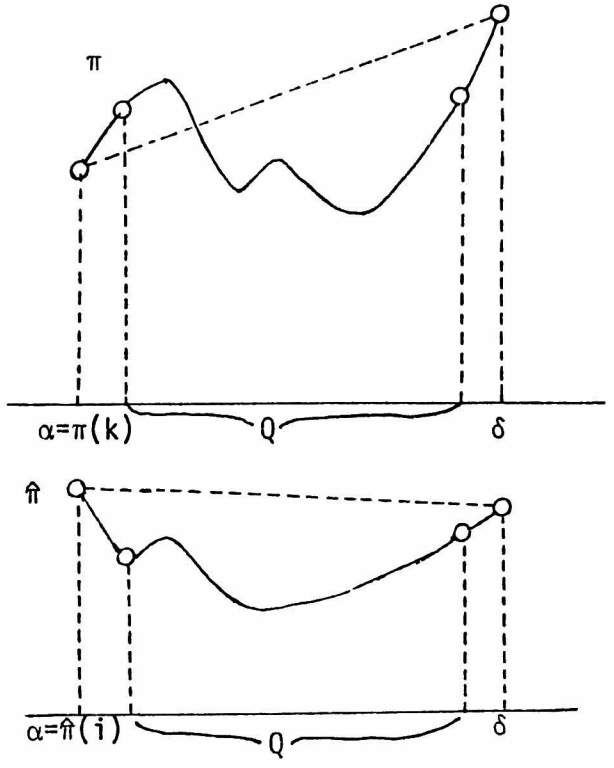


Figure 3.1. $\hat{\pi}$ with condition (a)(b)(c).

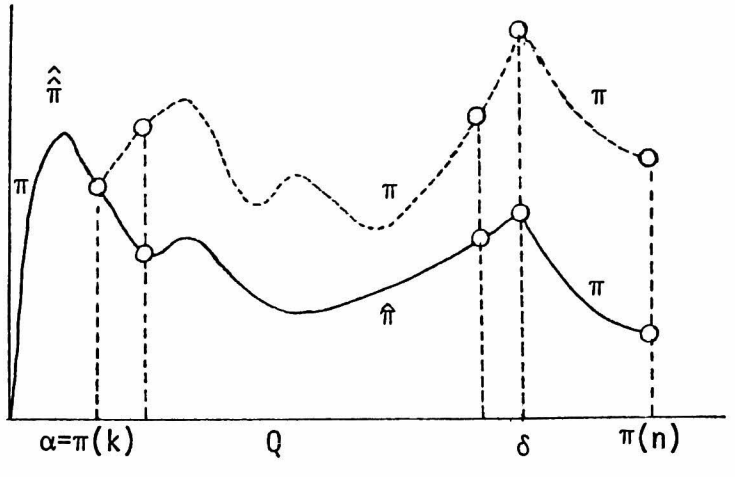


Figure 3.2. Illustration of construction method for $\hat{\hat{\pi}}$ in Proposition 3.3.

$$f_{\alpha}^{\hat{\hat{\pi}}} = f_{\alpha}^{\pi}, \qquad (3.2)$$

$$f_{\hat{\hat{\pi}}(k+j)}^{\hat{\hat{\pi}}} = f_{\delta}^{\hat{\hat{\pi}}} - f^{\hat{\hat{\pi}}}(\hat{\hat{\pi}}(k+j),\{\hat{\hat{\pi}}(k+j+1),\ldots,\hat{\hat{\pi}}(k+|Q|)\},\delta)$$

$$= f_{\delta}^{\hat{\hat{\pi}}} - f^{\hat{\hat{\pi}}}(\hat{\pi}(i+j),\{\hat{\pi}(i+j+1),\ldots,\hat{\pi}(i+|Q|)\},\delta)$$

$$\leqq f_{\delta}^{\hat{\hat{\pi}}}, \quad j=1,2,\ldots,|Q|, \text{ (by condition (c) of } \hat{\pi}) \qquad (3.3),$$

$$f_{\delta}^{\hat{\hat{\pi}}} = f_{\alpha}^{\hat{\hat{\pi}}} + f^{\hat{\pi}}(\alpha,Q,\delta) \leqq f_{\alpha}^{\hat{\hat{\pi}}} \text{ (by condition (b) of } \hat{\pi}) \qquad (3.4),$$

$$f_{\delta}^{\pi} \geqq f_{\pi(j)}^{\pi} = f_{\hat{\hat{\pi}}(j)}^{\hat{\hat{\pi}}}, \quad j=1,2,\ldots,k \text{ (since } \pi \in \Pi(\delta)) \qquad (3.5),$$

$$f_{\hat{\hat{\pi}}(k+|Q|+j)}^{\hat{\hat{\pi}}} - f_{\delta}^{\hat{\hat{\pi}}} = f_{\pi(k+|Q|+j)}^{\pi} - f_{\delta}^{\pi} \leqq 0, \quad j=1,2,\ldots,n-k-|Q|$$

(since $\pi \in \Pi(\delta)$) (3.6).

From these properties,

$$f_{\delta}^{\pi} \geqq f_{\alpha}^{\pi} = f_{\alpha}^{\hat{\hat{\pi}}} \geqq f_{\delta}^{\hat{\hat{\pi}}} \geqq f_{\hat{\hat{\pi}}(k+j)}^{\hat{\hat{\pi}}}, \quad j=1,2,\ldots,n-k, \qquad (3.7)$$

follows. By (3.1) and (3.7),

$$f^{\pi} = f_{\delta}^{\pi} \geqq \max_{k\in\{1,2,\ldots,n\}} f_{\hat{\hat{\pi}}(k)}^{\hat{\hat{\pi}}} = f^{\hat{\hat{\pi}}}$$

holds. Moreover, (3.7) implies

$$f_{\alpha}^{\hat{\hat{\pi}}} \geqq f_{\hat{\hat{\pi}}(k+j)}^{\hat{\hat{\pi}}}$$

and $\theta(\hat{\hat{\pi}})$ is either α or the vertex before α on $\hat{\pi}$. Thus $\hat{\hat{\pi}}$ constructed as above certainly satisfies the conditions in the proposition statement. □

Therefore we can exclude π from further consideration at P_δ if $\hat{\pi}$ satisfying the condition of Proposition 3.2 exists. This means that only $\hat{\Pi}(\delta)$ defined below need be considered instead of $\Pi(\delta)$.

$$\hat{\Pi}(\delta) \triangleq \{ \pi \in \Pi(\delta) \mid \text{any } \hat{\pi} \text{ satisfying condition (a)(b)(c) in Proposition 3.2 does not exist} \}$$

As a result of this property, the following $\hat{P}_\delta$ (instead of P_δ) is solved in order to find an optimal solution of B.

Problem $\hat{P}_\delta$: Find $\pi \in \hat{\Pi}(\delta)$ minimizing f_δ^π, i.e., the length from vertex 1 to vertex δ.

An optimal solution of $\hat{P}_\delta$ is also denoted by π_δ. Some more definitions are necessary to describe an algorithm for $\hat{P}_\delta$.

$$f(\alpha,Q,\beta) \triangleq min\{ f^\pi(\alpha,Q,\beta) \mid \pi \in \Pi(\delta) \cap \Pi(\alpha,Q,\beta) \} \quad (3.8)$$

where $Q \subset V - \{1,\alpha,\beta\}$, $\alpha \neq \delta$, $\delta \neq 1$.

$$g(\delta,Q,\gamma) \triangleq min\{ f^\pi(\delta,Q,\gamma) \mid \pi \in \Pi(\delta) \cap \Pi(\delta,Q,\gamma) \} \quad (3.9),$$

where $Q \subset V - \{1,\delta,\gamma\}$, $\gamma \neq \delta$, $\delta \neq 1$.

$$g(\delta,R) \underline{\Delta} min \{ g(\delta,Q,\gamma) \mid Q \cup \{\gamma\} = R\} \qquad (3.10).$$

Based on these notations, we are now ready to describe an algorithm for solving $\hat{P}_\delta$.

Algorithm for Problem $\hat{P}_\delta$

(I) (Calculation of $f(1,Q,\delta)$)

If $Q = \phi$, let

$$f(\alpha,\phi,\delta) = \begin{cases} c_{\alpha\delta} & (c_{\alpha\delta} > 0) \\ \infty & (c_{\alpha\delta} \leq 0) \end{cases} \qquad (3.11)$$

If $Q \neq \phi$, let

$$f(\alpha,Q,\delta) = \begin{cases} F_m & (F_m > 0) \\ \infty & (F_m \leq 0), \end{cases} \qquad (3.12)$$

where $F_m = \min_{\beta \in Q} (c_{\alpha\beta} + f(\beta, Q-\{\beta\}, \delta))$.

(II) (Calculation of $g(\delta,Q,\gamma)$)

If $Q = \phi$, let

$$g(\delta,\phi,\gamma) = \begin{cases} c_{\delta\gamma} & (c_{\delta\gamma} \leq 0) \\ \infty & (c_{\delta\gamma} > 0) \end{cases} \qquad (3.13)$$

If $Q \neq \phi$, let

$$g(\delta,Q,\gamma)=\begin{cases} G_m \quad (\ G_m \leqq 0) \\ \\ \infty \quad (\ G_m > 0)\ , \end{cases} \tag{3.14}$$

where $G_m = \underset{\beta \in Q}{min}\{\ c_{\beta\gamma} + g(\delta,Q-\{\beta\},\beta)\}$.

(III) (Calculation of π_δ)

$$f(1,Q^*,\delta) = min\ [\, g(1,Q,\delta) \mid g(\delta,\bar{Q}) \leqq 0,\ Q \subseteq V - \{1,\delta\}] \tag{3.15},$$

where $\bar{Q} \triangleq V - \{1,\delta\} - Q$.

Now we consider some further properties of $\theta(\pi)$, $\pi \in \overset{\sim}{\Pi}_n$. Let

$$I_p(j) \triangleq \{i \mid c_{ij} > 0,\ i \in V - \{j\}\},\quad j=2,3,\ldots,n,$$

$$I_N(j) \triangleq \{i \mid c_{ji} \leqq 0,\ i \in V - \{j\}\},\quad j=2,3,\ldots,n,$$

and

$$I_o \triangleq \{j \mid I_p(j) \neq \phi,\ I_N(j) \neq \phi\}.$$

Then the following Proposition 3.3 is obvious.

Proposition 3.3. For any $\pi \in \overset{\sim}{\Pi}_n$, $\theta(\pi) \in I_o$ or $\theta(\pi) = \pi(n)$ holds.

Proposition 3.3 shows that calculation of $f(1,V-\{\alpha,\delta\},\delta)$ is only needed for $\hat{P}_\delta$, $\delta \notin I_o$.

Now we return to the calculation of an optimal solution of $\hat{P}_\delta$.

Theorem 3.2. $\pi \in \Pi(\delta)$ giving

$$\begin{cases} \text{(a)} \quad f(1,Q^*,\delta) = f^\pi(1,Q^*,\delta) \\ \text{(b)} \quad g(\delta,\bar{Q}^*) = f^\pi(\delta,\bar{Q}^*-\{\gamma^*\},\gamma^*), \ \gamma^* \in \bar{Q}^* \end{cases}$$

for Q^* calculated in (III) above, is an optimal solution of $\hat{P}_\delta$.

Proof: (Correctness of the computation of $g(\delta,Q,\gamma)$ in II)

First note that, for $\pi \in \hat{\Pi}(\delta) \cap \Pi(\delta)$,

$$f^\pi(\delta,Q,\gamma) \leqq 0$$

must hold.

Case (i): $Q = \phi$. Since $f^\pi(\delta,\phi,\gamma) = c_{\delta\gamma}$ obviously holds, we have

$$g(\delta,\phi,\gamma) = c_{\delta\gamma}$$

if $c_{\delta\gamma} \leqq 0$ holds. On the other hand, if $c_{\delta\gamma} > 0$ holds, $\pi \notin \hat{\Pi}(\delta)$ holds. Thus $f(\delta,\sigma,\gamma)$ can be set to ∞ (i.e., it is excluded from further calculation) since

$$f^\pi(\delta,\phi,\gamma) > 0$$

holds for $\pi \in \Pi(\delta,\phi,\gamma)$.

Case (ii): $|Q| = s+1$. (Under the induction hypothesis that $g(\delta,Q,\gamma)$ correctly computed for $|Q| = s$). Let

$$G_m = \min_{\beta \in Q}(c_{\beta\gamma} + g(\delta,Q - \{\beta\},\beta)) \\ = c_{\beta^*\gamma} + g(\delta,Q - \{\beta^*\},\beta^*) \qquad (3.16)$$

Then two subcases are possible.

Subcase (iia): $G_m \leqq 0$. As g is set to either ∞ or $\leqq 0$ in this calculation,

$$g(\delta,Q - \{\beta^*\},\beta^*) \leqq 0$$

must hold. By the induction hypothesis, we obtain

$$g(\delta,Q - \{\beta^*\},\beta^*) = \min\{f^{\pi}(\delta,Q - \{\beta^*\},\beta^*) \mid \pi \in \hat{\Pi}(\delta) \\ \cap \Pi(\delta,Q - \{\beta^*\},\beta^*)\}. \qquad (3.17)$$

The principle of optimality that holds for this problem by an argument similar to [Bellman,'62b], together with (3.16) - (3.17), proves the validity of setting

$$g(\delta,Q,\gamma) = G_m .$$

In other word, this $g(\delta,Q,\gamma)$ is equivalent to $g(\delta,Q,\gamma)$ defined in (3.9).

Subcase (iib): $G_m > 0$. By the induction hypothesis,

$$\Pi(\delta,Q-\{\beta\},\beta)\cap\hat{\Pi}(\delta)=\phi$$

holds for β with $g(\delta,Q-\{\beta\},\beta)=\infty$. For β satisfying $g(\delta,Q-\{\beta\},\beta)\leqq 0$ & $c_{\beta\gamma}+g(\delta,Q-\{\beta\},\beta)>0$, and $\pi\in\Pi(\delta,Q-\{\beta\},\beta)\cap\Pi(\delta,Q,\gamma)$,

$$f^{\pi}(\delta,Q,\gamma)>0$$

holds. This implies that

$$\Pi(\delta,Q-\{\beta\},\beta)\cap\Pi(\delta,Q,\gamma)\cap\hat{\Pi}(\delta)=\phi.$$

In either case,

$$\hat{\Pi}(\delta)\cap\Pi(\delta,Q,\gamma)=\phi$$

is obvious. Therefore, it is justified to set $g(\delta,Q,\gamma)=\infty$.

(Validity of the calculation of $f(\alpha,Q,\delta)$ in (I) of the above algorithm)

From the condition (b) among the equivalent conditions to $\theta(\pi)$ noted in Section 3.3,

$$f^{\pi}(\alpha,Q,\delta)>0 \qquad (3.18)$$

holds for $\pi\in\hat{\Pi}(\delta)$.

Case (i): $Q=\phi$. First consider the case of $c_{\alpha\delta}>0$. It is clear that $f(\alpha,\phi,\delta)=c_{\alpha\delta}$. On the other hand, if $c_{\alpha\delta}\leqq 0$,

$\hat{\Pi}(\delta) \cap \Pi(\alpha,\phi,\delta) = \phi$ holds since $f^{\pi}(\alpha,\phi,\delta) \leq 0$ for any $\pi \in \Pi(\alpha,\phi,\delta)$. This proves $f(\alpha,\phi,\delta) = \infty$.

Case (ii): $|Q| = s+1$. (Under the induction hypothesis that $f(\alpha,Q,\delta)$ are correctly computed for $|Q| = s$). Let

$$F_m = \min_{\beta \in Q} \{ c_{\alpha\delta} + f(\beta, Q-\{\beta\}, \delta)\}$$
$$= c_{\alpha\beta^*} + f(\beta^*, Q-\{\beta^*\}, \delta). \tag{3.19}$$

Then three subcases are possible.

Case (iia): $\infty > F_m > 0$. Calculation in (I) of the above algorithm permits either $f = \infty$ or $0 < f < \infty$. Therefore

$$\infty > f(\beta^*, Q-\{\beta^*\}, \delta) > 0 \tag{3.20}$$

must hold. By the induction hypothesis,

$$f(\beta^*, Q-\{\beta^*\}, \delta) = \min\{f^{\pi}(\beta^*, Q-\{\beta^*\}, \delta) \mid \pi \in \hat{\Pi}(\delta) \cap \Pi(\beta^*, Q-\{\beta^*\}, \delta)\}$$

holds. By (3.19)-(3.21) and the principle of optimality of dynamic programming (see [Bellman,'62b]), it is possible to set

$$f(\alpha,Q,\delta) = F_m .$$

This is in fact equal to the f defined in (3.8).

Case (iib): $F_m = \infty$. Since $f(\beta,Q-\{\beta\},\delta) = \infty$ for any $\beta \in Q$, we have

$$\hat{\Pi}(\delta) \cap \Pi(\beta,Q-\{\beta\},\delta) = \phi$$

or

$$\hat{\Pi}(\delta) \cap \Pi(\alpha,Q,\delta) = \phi.$$

Therefore $f(\alpha,Q,\delta) = F_m = \infty$ follows.

Case (iic): $F_m \leqq 0$. From calculation in (I) of the above algorithm,

$$f(\beta^*,Q-\{\beta^*\},\delta) > 0$$

follows. Since

$$f(\beta^*,Q-\{\beta^*\},\delta) = min\{f^{\pi}(\beta^*,Q-\{\beta^*\},\delta) \mid \pi \in \hat{\Pi}(\delta) \cap \Pi(\beta^*,Q-\{\beta^*\},\delta)\}$$

holds by the induction hypothesis, we have

$$f(\tilde{\pi}(i+j),\{\tilde{\pi}(i+j+1),\ldots,\tilde{\pi}(i+|Q|)\},\delta) > 0$$

for all $j=1,2,\ldots,|Q|$, if $\tilde{\pi}(i+1) = \beta^*$ holds for $\tilde{\pi}$ giving this $f(\beta^*,Q-\{\beta^*\},\delta)$. Construct $\hat{\pi}$ from $\tilde{\pi}$ as follws:

$$\hat{\pi}(i) = \alpha, \quad \hat{\pi}(i+j) = \tilde{\pi}(i+j), \quad (j=1,2,\ldots,|Q|).$$

This $\hat{\pi}$ satisfies conditions (a)(b)(c) in Proposition 3.2.

Therefore, by Proposition 3.2,

$$\hat{\Pi}(\delta) \cap \Pi(\alpha, Q, \delta) = \phi$$

holds. Thus it is possible to set

$$f(\alpha, Q, \delta) = \infty.$$

(Correctness of the computation of (III))
By the definition of $g(\delta, Q, \gamma)$, $g(\delta, \bar{Q}) \leqq 0$ implies that there exists $\pi \in \hat{\Pi}(\delta) \cap \Pi(\delta, \bar{Q}-\{\gamma\}, \gamma)$ for some $\gamma \in \bar{Q}$. Therefore (3.8) and (3.9) together shows that π defined by

$$f^{\pi}(1, Q^*, \delta) = f(1, Q^*, \delta)$$

is an optimal solution of $\hat{P}_{\delta}$. □

3,5 Solution Procedure for Problem B

As shown in Theorem 3.2, the above algorithm which consists of three parts (I)(II)(III), solves each $\hat{P}_\delta$. If in the course of calculation it is shown by Proposition 3.3 below or other reasons that solutions better than the current best solution will not be found from those derived (α,Q,β), the calculation with respect to all $(\gamma,\hat{Q},\delta)$ for $\hat{Q}\supset Q\cup\{\alpha\}$ can be terminated.

Proposition 3.3. Let an upper bound of the optimal value of Problem B (say, the value of the current best solution $\bar{\pi}$) be denoted by $\bar{f}$. Then in the course of calculation with respect to $\hat{P}_\delta$, all $f(\alpha,Q,\delta)$ with $\infty > F_m \geqq \bar{f}$, can be set to ∞. (That is, any $\pi\in\hat{\Pi}(\delta)\cap\Pi(\alpha,Q,\delta)$ cannot be a better solution than $\bar{\pi}$.

Proof: Case (i): $\alpha=1$. In calculating (3.11), $f(1,Q,\delta)$ is set to F_m. As described in the Proof of Theorem 3.2,

$$f^\pi = f^\pi(1,Q,\delta) \geqq f(1,Q,\delta) > \bar{f} \tag{3.23}$$

holds for all $\pi\in\Pi(\delta)\cap\Pi(1,Q,\delta)$. Therefore $f(1,Q,\delta)$ can be set to ∞.

Case (ii): $\alpha\neq 1$. For all $\pi\in\Pi(\alpha,Q,\delta)$,

$$f_\delta^\pi = f_\alpha^\pi + f^\pi(\alpha,Q,\delta) \tag{3.24}$$

holds. Now, if $\alpha=\pi(i)$ holds, we have

$$f_{\alpha}^{\pi} = \sum_{k=1}^{i-1} c_{\pi(k)\pi(k+1)} = \sum_{k=1}^{i-1} c_{\pi(k)\pi(k+1)} + t_{\alpha} > 0 \quad (3.25).$$

By (3.24) and (3.25), this implies that

$$f_{\delta}^{\pi} \geq f^{\pi}(\alpha,Q,\delta).$$

Especially, for all $\pi \in \Pi(\alpha,Q,\delta) \cap \hat{\Pi}(\delta)$, we have

$$f^{\pi} = f_{\delta}^{\pi} \geq f^{\pi}(\alpha,Q,\delta) \geq \bar{f} \qquad (3.26).$$

Since (3.26) implies that π cannot give a better solution than $\bar{\pi}$, it is possible to set $f(\alpha,Q,\delta)$ to ∞. □

Based on Proposition 3.3, we can give an algorithm for solving Problem B.

Algorithm for Problem B

Phase I: Let $\bar{\pi}(i) = i$, $i=1,2,\ldots,n$, be the initial current best solution, and let f^{π} be an upper bound of $\bar{f}$.

Phase II: Decomposing Problem B into subproblems $\hat{P}_2,\hat{P}_3,\ldots,\hat{P}_n$, we solve $\hat{P}_2,\ldots,\hat{P}_n$ in this order. Each subproblem $\hat{P}_{\delta}$ is solved as given below: Calculate $f(\alpha,Q,\delta)$ in increasing order of $k = |Q|$. Note, however, that calculating $f(1,Q,\delta)$ is executed for only $Q \supseteq V(\delta)$ (see Figure 3.1). In the course of calculating $f(\alpha,Q,\delta)$, $g(\delta,\bar{Q})$ is obtained as soon as $f(1,Q,\delta)$ with $0 < f(1,Q,\delta) < \bar{f}$ and $Q \subseteq V-\{1,\delta\}$ is calculated. If $g(\delta,\bar{Q}) < 0$, π is stored as $\bar{\pi}$ and $\bar{f}$ is set to f^{π} where π is obtained from the compu-

tational result of $f(1,Q,\delta)$ and $g(\delta,\bar{Q})$. Calculation of $\hat{P}_\delta$ is continued until $f(1,V-\{1,\delta\},\delta)$ is calculated. If $f(\alpha,Q,\delta)=\infty$ holds for all Q with $|Q|=k$, then the calculation of $\hat{P}_\delta$ halts at this point, because an optimal solution of Problem B cannot be found from the further calculation of $\hat{P}_\delta$.

Phase III: Upon completion of solving P_n, the current $\bar{\pi}$ and $\bar{f}$ are an optimal solution and the optimal value respectively.

3.6 Example

Consider the four city problem given in Table 3.1 and Table 3.2 with $t_1=0$, $t_2=33$, $t_3=17$, $t_4=15$. (Note that $V(2)=\{1\}$, $V(3)=\{1,2\}$, $V(4)=\{1,2,3\}$ by Proposition3.1.)

Table 3.1. t_{ij}(time from city i to city j, $i\neq j$, $1\leqq i\leqq n$, $1\leqq j\leqq n$).

i \ j	1	2	3	4
1		4	1	6
2	6		9	3
3	6	2		1
4	3	2	5	

Table 3.2. $C_{ij}=t_{ij}+t_j-t_i$ $(1\leqq i\leqq n,\ 1\leqq j\leqq n,\ i\neq j)$.

i \ j	1	2	3	4
1		37	18	21
2	-27		-7	-15
3	-11	18		-1
4	-12	20	7	

(I) Initial setting: $\bar{\pi}(1) = 1$, $\bar{\pi}(2) = 2$, $\bar{\pi}(3) = 3$, $\bar{\pi}(4) = 4$,

$$\bar{f} = f^{\bar{\pi}} = max(f_2^{\bar{\pi}}, f_3^{\bar{\pi}}, f_4^{\bar{\pi}}) = max(c_{12}, c_{12} + c_{23}, c_{12} + c_{23} + c_{34}) = max(37, 37 + (-7) = 30, 37 + (-7) + 1 = 31) = 37.$$

(II) $\hat{P}_2$: Case of $k = |Q| = 0$: Though $f(1,\phi,2) = c_{12} = 37$ holds, it is set to

$$f(1,\phi,2) = \infty$$

by Proposition 3.3 since $37 \leq \bar{f}$. As for $f(3,\phi,2)$ and $f(4,\phi,2)$,

$$f(3,\phi,2) = c_{32} = 18 \ (< \bar{f}), \quad f(4,\phi,2) = c_{42} = 20 \ (< \bar{f}).$$

Case of $k = |Q| = 1$:

$$f(1,\{3\},2) = F_m = c_{13} + f(3,\phi,2) = 18 + 18 = 36 \ (< \bar{f}).$$

$$f(1,\{4\},2) = \infty \ (F_m = c_{14} + f(4,\phi,2) = 21 + 20 = 41 \geq \bar{f}).$$

Since $f(1,\{3\},2) < \infty$, $g(2,\{4\})$ is calculated. As $|Q| = 1$ and $\bar{Q} = \{4\}$, only $g(2,\phi,4)$ need be calculated.

$$g(2,\phi,4) = c_{24} = -c_{15} \ (< 0).$$

The solution corresponding to $f(1,\{3\},2)$ and $g(2,\phi,4)$ is better than current $\bar{\pi}$. Therefore the new $\bar{\pi}$ and $\bar{f}$ become as follows:

$\bar{\pi}$: $\pi(1) = 1,\ \pi(2) = 3,\ \pi(3) = 2,\ \pi(4) = 4.$

$\bar{f} = 36.$

We proceed to further calculations of $k = |Q| = 1$.

$$f(3,\{4\},2) = c_{34} + f(4,\phi,2) = (-1) + 20 = 19\ (< \bar{f}\ ).$$

$$f(4,\{3\},2) = c_{43} + f(3,\phi,2) = 7 + 18 = 25\ (< \bar{f}\ ).$$

Case of $k = |Q| = 2$:

$$f(1,\{3,4\},2) = \infty\ (\ F_m = min\{c_{13} + f(3,\{4\},2),\ c_{14} + f(4,\{3\},2)\} = min\{18 + 19 = 37,\ 21 + 25 = 46\}$$

$$= 37\ (> \bar{f}).$$

Calculation of $\hat{P}_2$ is completed.

$\hat{P}_3$: Case of $k = |Q| = 0$: As $|V(3)|-1 = 1 > k$, $f(1,\phi,3)$ need not be calculated by Proposition 3.1.

$$f(2,\phi,3) = \infty\ (\ c_{23} = -7 < 0,\ \text{so by Proposition 3.2}).$$

$$f(4,\phi,3) = c_{43} = 7\ (< \bar{f}\ ).$$

Case of $k = |Q| = 1$: As $Q \supseteq V(3)-\{1\} = \{2\}$ by Proposition 3.1, only $f(1,\{2\},3)$ is calculated for $\alpha \neq 1$.

$$f(1,\{2\},3) = c_{12} + f(2,\phi,4) = \infty.$$

Since

$$f(2,\{4\},3) = \infty \left(F_m = c_{24} + f(2,\phi,3) = -15 + 7 \leq 0 \text{ and} \right.$$

by Proposition 3.2),

$$f(4,\{2\},3) = c_{42} + f(2,\phi,3) = \infty,$$

and hence

$$f(\alpha,Q,\delta) = \infty$$

for all $\alpha \neq 1$, $Q \subseteq V-\{1,\delta\}$, $k = |Q|$, the calculation of $\hat{P}_3$ is completed.

$\hat{P}_4$: Case of $k = |Q| = 0$: As $|V(4)|-1 = 2 > k = 0$, only $f(3,\phi,4)$ and $f(2,\phi,4)$ are calculated.

$$f(3,\phi,4) = \infty \ (\ c_{34} = -1 < 0 \text{ and apply Proposition 3.2}).$$

$$f(4,\phi,2) = \infty \ (\ c_{24} = -15 < 0 \text{ and apply Proposition 3.2}).$$

For all $\alpha \in V-\{1,4\}$, we have $f(2,\phi,4)=\infty$. Thus the calculation of $\hat{P}_4$ is completed.

(III) At this point, all subproblems $\hat{P}_2$, $\hat{P}_3$ and $\hat{P}_4$ are solved. An optimal solution is π^* ($\pi^*(1)=\bar{\pi}(1)=1$, $\pi^*(2)=\bar{\pi}(2)=3$, $\pi^*(3)=\bar{\pi}(3)=2$, $\pi^*(4)=\bar{\pi}(4)=4$) and the optimal value is $f^{\pi^*}=\bar{f}=36$ (this solution is illustrated in Figure 3.3). Note that the completion time of the production at $\pi^*(3)=2$ (the 3rd visited city) is the latest. This exemplifies that an optimal solution π ($\pi(1)=1$, $\pi(2)=3$, $\pi(3)=4$, $\pi(4)=2$) of the ordinary traveling salesman problem with the objective function

$$\sum_{i=1}^{n-1} t_{\pi(i)\pi(i+1)}$$

is not necessarily an optimal solution of the minimax type traveling salesman problem.

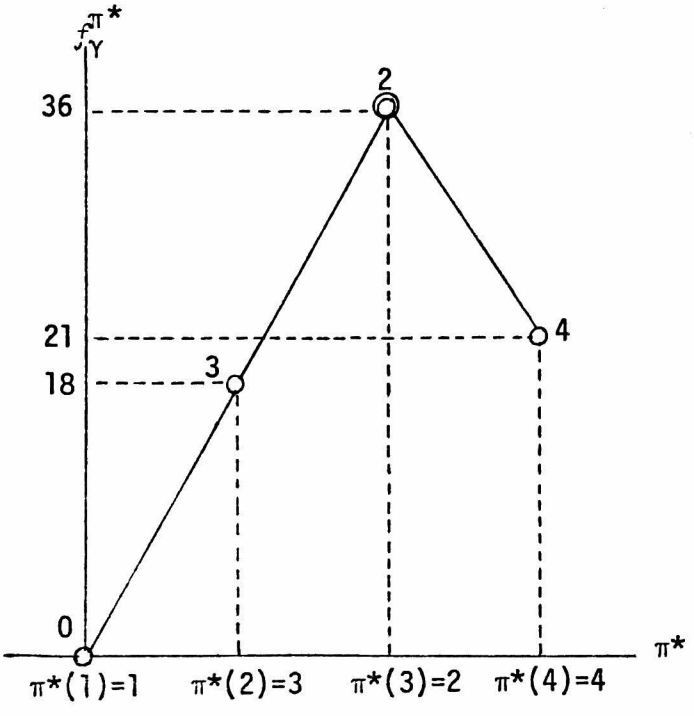


Figure 3.3. An optimal solution π^*.

3.7 Conclusion

Although a direct adaptation of dynamic programming approach for solving Problem B may be difficult, the decomposition into subproblems $P_2, P_3, \ldots, P_n$ enables us to use dynamic programming procedure as shown in Section 3.3 and Section 3.4. Some further properties useful to reduce computational efforts for Problem B are also given. Nevertheless, the computational complexity of this algorithm is $O(n^3 \times 2^{n-3})$ in the worst case. Although this order is much less than the order $(n-1)!$ of the complete enumeration of all permutations, the proposed solution procedure will become impractical for large n. The development of more efficient procedures is a future research problem.

CHAPTER 4 TRAVELING SALESMAN PROBLEMS WITH A CAPACITY CONSTRAINT OF THE DELIVERY TRUCK

4.1 Introduction

This chapter discusses the ordinary traveling salesman problem and the delivery route problem treated in Chapter 3 with an additional constraint that the truck can carry only the amount not exceeding its loading capacity. Similarly to the previous problem in Section 3, the truck leaves the starting city with a certain amount of resources loaded, delivers some amount of resources to each city and at the same time, collects some amount of production wastes. At each city i, the total sum of the remaining resources and the collected wastes cannot exceed the capacity of the truck.

As an example, let us consider the following milk delivery problem: A truck delivers boxes containing some milk bottles from the plant to the milk shop at each city i and at the same time collects empty bottles. On receiving the milk boxes from the truck, the milk man of each shop delivers milk bottles to his consumers. Moreover, the truck has a loading capacity and the total number of boxes with packed bottles and empty bottles cannot exceed this capacity. The objective is to minimize the time of completing the milk delivery to all the consumers in all the cities.

Because of the additional constraint, most of the algorithms developed so far for the ordinary traveling salesman problem cannot be directly applied to the new

problem. The purpose of this chapter is to extend two existing dynamic programming algorithms to the present problem. The first is the dynamic programming algorithm proposed by Held and Karp [Held and Karp,'62] and Bellman [Bellman,'62b] for the ordinary traveling salesman problem, and the latter is the one discussed in Chapter 3 for the minimax type traveling salesman problem.

Section 4.2 describes a generalization of the dynamic programming algorithms by Held and Karp, and Bellman so that the capacity constraint is incorporated. Section 4.3-4.6 treat the minimax type traveling salesman problem. Section 4.3 introduces a formulation of the minimax type traveling salesman problem $\bar{B}$ with the capacity constraint. In Section 4.4, Problem $\bar{B}$ is decomposed into subproblems according to the visiting order of the truck. Section 4.5 provides an algorithm for Problem $\bar{B}$ which is obtained by combining the dynamic programming algorithms for the generated subproblems. The dynamic programming algorithms for subproblems are slightly different from the ones discussed in Chapter 3. This section also includes the proof for the validity of the proposed algorithm. Section 4.6 gives an illustrative example. Finally, Section 4.7 is a summary of this chapter.

4.2 Modification of the Held-Karp-Bellman Algorithm

First, the leading capacity of the delivery truck is assumed to be Δ. It leaves the starting city 1 and delivers r_i amount of production resources to each city i, $i=2,3,\ldots,n$ and at the same time collects w_i amount of production wastes. For the purpose of the formulation, the following graph theoretic notations are introduced. Let $G=(V,E)$ be the complete directed graph with vertex set $V=\{v_1,v_2,\ldots,v_n\}$ and arc set $E=\{(v_i,v_j) \mid v_i \in V,\ v_j \in V,\ i\neq j\}$ same as the one introduced in Chapter 3. In this graph, a vertex corresponds to a city and an arc to a road, respectively. Let c_{ij} be the length of arc (i,j). From the definition of the traveling salesman problem, it is sufficient to consider a hamiltonian path which starts from vertex 1 and visits every vertex exactly once. The length of the hamiltonian path is the sum of the length of the arcs therein.

In general, it is assumed that at each city i, the production is initiated upon receiving the production resources and takes t_i unit times until the completion. First, this section treats the case in which the production time is neglected, i.e., $t_i=0$, $i=1,2,\ldots,n$. By the capacity constraint, it is required that for a hamiltonian path $\pi=(\pi(1)(=1),\ldots,\pi(n))$,

$$\sum_{i=2}^{k} w_{\pi(i)} + \sum_{i=k+1}^{n} r_{\pi(i)} \leqq \Delta \qquad (4.1)$$

holds for $k=2,3,\ldots,n$. Thus the problem is to find a

hamiltonian path that has the minimum length among those satisfying constraint (4.1).

Letting

$$\Delta_i = r_i - w_i \ , \quad i=2,3,\ldots,n,$$

$$\Delta_1 = \Delta - \sum_{i=2}^{n} r_i \ ,$$

(4.1) is transformed into

$$\min_{k=2,3,\ldots,n} \sum_{i=1}^{k} \Delta_{\pi(i)} \geq 0 \qquad (4.2)$$

(Note that $\sum_{i=1}^{k} \Delta_{\pi(i)}$ denotes the margin of the truck capacity at city $\pi(k)$).

Now let Π_n denote the set of permutations on V as defined in Chapter 3 and let

$$\bar{\Pi}_n \triangleq \{ \pi \in \Pi_n \mid \pi(1)=1 \text{ and } \pi \text{ satisfies (4.2)} \}.$$

For $Q \subset V-\{1\}$ and $\alpha,\beta \in V-Q$, $\bar{\Pi}(\alpha,Q,\beta)$ denotes the subset of $\bar{\Pi}_n$ such that

$$\begin{cases} \text{(a)} \quad \pi(i) = \alpha \\ \text{(b)} \quad \pi(i+j) \in Q, \quad j=2,3,\ldots,|Q| \\ \text{(c)} \quad \pi(i+|Q|+1) = \beta \end{cases}$$

for some i. With these preparations, $f(1,Q,\beta)$ and $\Delta(\alpha,Q,\beta)$ are defined by

$$f(1,Q,\beta) = \min_{\pi}\{\sum_{i=1}^{|Q|+1} c_{\pi(i)\pi(i+1)} \mid \pi \in \bar{\Pi}(1,Q,\beta)\}$$

$$\Delta(\alpha,Q,\beta) = \Delta_{\alpha} + \sum_{i \in Q} \Delta_i + \Delta_{\beta} \quad .$$

Obviously, the present problem is solved if $f(1,V-\{1,\beta\},\beta)$ are obtained for all $\beta \in V-\{1\}$. The optimal value is given by

$$\min_{\beta}\{f(1,V-\{1,\beta\},\beta) \mid \beta \in V-\{1\}\}.$$

Each $f(1,V-\{1,\beta\},\beta)$ is calculated by the following recursion.

Algorithm for calculating $f(1,V-\{1,\beta\},\beta)$

Step 1 ($Q=\phi$): For $\beta \in V-\{1\}$, let

$$\Delta(1,\phi,\beta) = \Delta_1 + \Delta_{\beta}$$

and

$$f(1,\phi,\beta) = \begin{cases} c_{1\beta} & (\ \Delta(1,\phi,\beta) \geqq 0\ ) \\ \infty & (\text{otherwise}) \end{cases}$$

Step 2 ($|Q| \geqq 1$): Let

$$\Delta(1,Q,\beta) = \Delta(1,Q-\{\alpha\},\alpha) + \Delta_{\beta} \text{ for any } \alpha \in Q,$$

$$f(1,Q,\beta)=\begin{cases} \min_{\beta\in Q}\{f(1,Q-\{\alpha\},\alpha)+c_{\alpha\beta}\} & (\ \Delta(1,Q,\beta)\geqq 0\ ) \\ \\ \infty \quad \text{(otherwise)} . \end{cases}$$

Step 2 is executed for all $Q\subset V-\{1\}$ in the non-decreasing order of $|Q|$. Upon completion of Step 2, $f(1,V-\{1,\beta\},\beta)$ are calculated for all $\beta\in V-\{1\}$.

The validity of this algorithm may be proved by the principle of optimality of dynamic programming in a manner similar to [Held and Karp,'62]. The difference from [Held and Karp,'62] is that the above recursion algorithm takes into account $\Delta(1,Q,\beta)$ (the margin of the truck capacity when it gets to vertex β), and whenever the capacity is exceeded (i.e., $\Delta(1,Q,\beta)<0$), the corresponding path is abandoned ($f(1,Q,\beta)$ is set to ∞). This is possible since all paths in $\bar{\Pi}(1,Q,\beta)$ have the same margin $\Delta(1,Q,\beta)$ at vertex β.

4.3 Formulation of the Minimax Traveling Salesman Problem with a Capacity Constraint

This section treats the general case with $t_i \geq 0$, $i=2,3,\ldots,n$. Rearranging the city number i ($i=2,3,\ldots,n$) and setting $t_1 = 0$, we can assume $t_2 \geq \cdots \geq t_n \geq 0$ without loss of generality.

Given $\pi \in \bar{\Pi}_n$, the completion time of production at vertex $\pi(k)$ is

$$\sum_{i=1}^{k-1} t_{\pi(i)\pi(i+1)} + t_{\pi(k)}$$

and the completion at all vertices is

$$f^{\pi} = max\{ \sum_{i=1}^{k-1} t_{\pi(i)\pi(i+1)} + t_{\pi(k)} \mid k=2,3,\ldots,n\}.$$

The above problem is first formulated as the following Problem $\bar{A}$.

Problem $\bar{A}$: Find $\pi \in \bar{\Pi}_n$ minimizing f^{π}.

For the complete directed graph $G=(V,E)$ introduced in Section 4.2, adjoin length c_{ij} to each arc (i,j) and capacity Δ_β to each vertex β where c_{ij} is defined by

$$c_{ij} = t_{ij} + t_j - t_i \quad (1 \leq i \leq n,\ 1 \leq j \leq n,\ i \neq j).$$

Then, for a hamiltonian path $\pi \in \bar{\Pi}_n$,

$$f^{\pi}_{\pi(k)} = \sum_{i=1}^{k-1} c_{\pi(i)\pi(i+1)}$$

denotes the length from vertex 1 (=$\pi(1)$) to vertex $\pi(k)$. Since

$$\sum_{i=1}^{k-1} t_{\pi(i)\pi(i+1)} + t_{\pi(k)}$$

$$= \sum_{i=1}^{k-1} (t_{\pi(i)\pi(i+1)} + t_{\pi(i+1)} - t_{\pi(i)}) - t_{\pi(1)}$$

$$= \sum_{i=1}^{k-1} c_{\pi(i)\pi(i+1)} \quad (t_{\pi(1)} = t_1 = 0)$$

$$= f^{\pi}_{\pi(k)} \ ,$$

Problem $\bar{A}$ is equivalent to the problem $\bar{B}$ below.

Problem $\bar{B}$: Find a hamiltonian path $\pi \in \bar{\Pi}_n$ minimizing

$$max\{f^{\pi}_{\pi(k)} | k=2,3,\ldots,n\}.$$

$\bar{B}$ is the same minimax type traveling salesman problem as the one discussed in the previous Chapter 3 except that the present problem has the capacity constraint.

4.4 Decomposition of Problem $\bar{B}$ into Subproblems $\bar{P}_2,\cdots,\bar{P}_n$

For $\pi \in \bar{\Pi}(\alpha,Q,\beta)$ defined in Section 4.2, $f^{\pi}(\alpha,Q,\beta)$ denotes the length of the portion from vertex α to vertex β of path π, i.e.,

$$f(\alpha,Q,\beta) = \sum_{k=i}^{i+|Q|} c_{\pi(k)\pi(k+1)} ,$$

where $\pi(i) = \alpha$. $k^*(\pi)$ denotes the smallest k^* satisfying

$$\max_k \{f^{\pi}_{\pi(k)} \mid k=2,3,\ldots,n\} = f^{\pi}_{\pi(k^*)}.$$

Define $\bar{\Pi}(\ell)$ by

$$\bar{\Pi}(\ell) = \{\pi \in \bar{\Pi}_n \mid k^*(\pi)=\ell\}$$

(i.e., $\pi \in \bar{\Pi}(\ell)$ satisfies $f^{\pi}_{\pi(\ell)} \geq f^{\pi}_{\pi(k)}$ for $k=2,3,\ldots,n$).

Problem $\bar{B}$ is now decomposed into the following (n-1) subproblems $\bar{P}_2,\cdots,\bar{P}_n$.

Problem $\bar{P}_\ell$: Find a hamiltonian path $\pi = \pi_\ell \in \bar{\Pi}(\ell)$ minimizing $f^{\pi}_{\pi(\ell)}$. ($f^{\pi_\ell}_{\pi_\ell(\ell)}$ is hereafter denoted simply as f_ℓ).

Theorem 4.1. Let $\pi_2,\pi_3,\cdots,\pi_n$ be optimal solutions of $\bar{P}_2,\cdots,\bar{P}_n$ respectively. An optimal solution of $\bar{B}$ is π_{ℓ^*}, where

$$f_{\ell^*} = \min\{f_\ell \mid \ell=2,3,\ldots,n\}.$$

Proof: $\bar{B}$ asks to calculate

$$\min_{\pi\in\bar{\Pi}_n} \max_k \{f^{\pi}_{\pi(k)} \mid k=2,3,\ldots,n\}$$

$$= \min_{2\leq\ell\leq n} \min_{\pi\in\bar{\Pi}(\ell)} \max_k \{f^{\pi}_{\pi(k)} \mid k=2,3,\ldots,n\} \quad (\bar{\Pi}_n = \bigcup_{\ell=2}^{n} \bar{\Pi}(\ell))$$

$$= \min_{2\leq\ell\leq n} \min_{\pi\in\bar{\Pi}(\ell)} f^{\pi}_{\pi(\ell)} = \min_{2\leq\ell\leq n} f_\ell \ .$$

f_ℓ is obtained by solving $\bar{P}_\ell$. □

4.5 An Algorithm for Solving Problem $\bar{B}$

Now an algorithm based on Theorem 4.1 is given for solving Problem $\bar{B}$. It solves $\bar{P}_2, \bar{P}_3, \cdots, \bar{P}_n$ in this order. In the algorithm, d* denotes the current best value of f^{π} (initially set to ∞). For $Q \subset V-\{1\}$, such that $|Q| < \ell-2$, $f^{\ell}(1,Q,\beta)$ is equal to the minimum length at vertex β for those paths which start from vertex 1, pass through all the vertices in Q and reach β, provided that it is smaller than d*; otherwise it is set to ∞. $q(1,Q,\beta)$ denotes the path giving $f^{\ell}(1,Q,\beta)$ (i.e., it starts from vertex 1, passes through the vertices in Q, and reaches vertex β). $M(1,Q,\beta)$ denotes the maximum length attained by a vertex in $(1,Q,\beta)$.

For Q with $|Q| = \ell-2$, $f^{\ell}(1,Q,\beta)$ $(\beta = \delta \leq \ell)$ has the same meaning as above, if the path corresponding to $f^{\ell}(1,Q,\beta)$ $(\beta = \delta)$ satisfies

$$f^{\pi}_{\pi(\ell)} > f^{\pi}_{\pi(k)} \quad k=1,2,\ldots,\ell-1.$$

If otherwise, it is again set to ∞ (since $\pi \notin \bar{\Pi}(\ell)$ necessarily holds for any $\pi \in \bar{\Pi}(1,Q,\delta)$). $q(1,Q,\delta)$ has the same meaning as above. $q(\delta,\tilde{Q},\gamma)$ and $q(\delta,\bar{Q})$ in Step 2 of Phase II denote the path giving $f^{\ell}(\delta,\tilde{Q},\gamma)$ and $f^{\ell}(\delta,\bar{Q})$ respectively where $\bar{Q} = V-\{1,\delta\}-Q$, $\tilde{Q} \subset \bar{Q}$ and $\gamma \in Q-\tilde{Q}$.

Algorithm for solving Problem $\bar{B}$

Phase I:

Step 1: Let $\ell \leftarrow 2$, d* $\leftarrow \infty$ and go to Phase II.

Step 2: If $\ell = n$, terminate. π^* is the optimal solution

of $\bar{B}$ and d^* is its value; otherwise, go to Phase II after $\ell \leftarrow \ell + 1$.

Phase II: (Solve $\bar{P}_\ell$)

Step 1: (Calculate $f^\ell(1,Q,\beta)$ for $|Q| \leqq \ell-2$ and $\beta \in V-\{1\}-Q$)

If $\ell = 2$, go to Substep 1c; otherwise go to Substep 1a.

Substep 1a: For $Q = \phi$ and $\beta \in V-\{1\}$, let

$$\Delta(1,\phi,\delta) \leftarrow \Delta_1 + \Delta_\beta,$$

$$f(1,\phi,\beta) \leftarrow \begin{cases} c_{1\beta} & (\Delta(1,\phi,\beta) \geqq 0 \text{ and } c_{1\beta} < d^*) \\ \infty & (\text{otherwise}), \end{cases}$$

$$M(1,\phi,\beta) \leftarrow f^\ell(1,\phi,\beta)$$

and

$$q(1,\phi,\beta) \leftarrow (1,\beta) \text{ if } f^\ell(1,\phi,\beta) < \infty.$$

Go to Substep 1b.

Substep 1b: For Q with $1 \leqq |Q| \leqq \ell-3$ and $\beta \in V-\{1\}-Q$, let (in the non-decreasing order of Q)

$$\Delta(1,Q,\beta) \leftarrow \Delta(1,Q-\{\alpha\},\alpha) + \Delta_\beta \text{ for any } \alpha \in Q,$$

$$f^\ell(1,Q,\beta) \leftarrow \begin{cases} F_m & (\Delta(1,Q,\beta) \geqq 0 \text{ and } F_m < d^*) \\ \infty & (\text{otherwise}), \end{cases}$$

$$M(1,Q,\beta) \leftarrow max(f^\ell(1,Q,\beta),\ M(1,Q-\{\alpha^*\},\alpha^*)),$$

and

$$q(1,Q,\beta) \leftarrow (q(1,Q-\{\alpha^*\},\alpha^*),\beta) \text{ if } f^{\ell}(1,Q,\beta) < \infty,$$

where

$$F_m = \min_{\alpha \in Q} \{ c_{\alpha\beta} + f^{\ell}(1,Q-\{\alpha\},\alpha)\}$$

$$= c_{\alpha^*\beta} + f^{\ell}(1,Q-\{\alpha^*\},\alpha^*).$$

Substep 1c†: For Q with $|Q| = \ell-2$ and $\delta \in V-\{1\}-Q$ with $\delta \leqq \ell$, let

$$\Delta(1,Q,\delta) \leftarrow \Delta(1,Q-\{\alpha\},\alpha) + \Delta_{\delta} \text{ for any } \alpha \in Q;$$

$$f^{\ell}(1,Q,\delta) \leftarrow \begin{cases} F_m & (\Delta(1,Q,\delta) \geqq 0 \text{ and } d^* > F_m > M(1,Q-\{\alpha^*\},\alpha^*)) \\ \infty & (\text{otherwise}), \end{cases}$$

and

$$q(1,Q,\delta) \leftarrow (q(1,Q-\{\alpha^*\},\alpha^*),\delta) \text{ if } f^{\ell}(1,Q,\delta) < \infty,$$

where

$$F_m = \min_{\alpha \in Q} \{c_{\alpha\beta} + f^{\ell}(1,Q-\{\alpha\},\alpha)\}$$

$$= c_{\alpha^*\beta} + f^{\ell}(1,Q-\{\alpha^*\},\alpha^*).$$

Step 2: (Test of the feasibility of each $f^{\ell}(1,Q,\delta)$)††For

---

†For $\delta > \ell$, $f^{\ell}(1,Q,\delta)$ cannot satisfy $f^{\ell}(\delta,\bar{Q}) \leqq 0$. For details, see $V(\delta)$ defined in previous Chapter 3.

††This step tests whether the path corresponding to $f^{\ell}(1,Q,\delta)$(i.e., $q(1,Q,\delta)$) obtained in Step 1c can be completed by attaching the last portion (i.e., $q(\delta,\bar{Q})$) so that the capacity constraint is satisfied, and the resulting path still has the maximum at δ. This completion is possible if and only if $f^{\ell}(\delta,\bar{Q}) \leqq 0$ holds.

each $f^{\ell}(1,Q,\delta) < \infty$ obtained in Step 1c, let $\bar{Q} \leftarrow V-Q-\{1,\delta\}$. If $\bar{Q}=\phi$, go to Step 3; otherwise go to Substeps 2a-2c, and obtain $f^{\ell}(\delta,\tilde{Q},\gamma)$ for each $\tilde{Q}$ and γ such that $\tilde{Q} \subset \bar{Q}$ and $\gamma \in \bar{Q} - \tilde{Q}$.

Substep 2a: For $Q = \phi$ and $\gamma \in \bar{Q}(= V-\{1,\delta\}-Q)$, let

$$\Delta(\delta,\phi,\gamma) \leftarrow \Delta_{\delta} + \Delta_{\gamma},$$

$$f^{\ell}(\delta,\phi,\gamma) \leftarrow \begin{cases} c_{\delta\gamma} \ (\Delta(1,Q,\delta)+\Delta(\delta,\phi,\gamma)-\Delta_{\delta}(=\Delta(1,Q \cup \{\delta\},\gamma)) \\ \quad \geq 0 \text{ and } c_{\delta\gamma} \leq 0) \\ \infty \ \text{(otherwise)}, \end{cases}$$

and

$$q(\delta,\phi,\gamma) \leftarrow (\gamma) \text{ if } f^{\ell}(\delta,\phi,\gamma) < \infty.$$

Substep 2b: For each $\tilde{Q} \subset \bar{Q}$ with $\tilde{Q} \neq \phi$ and $\gamma \in \bar{Q}-\tilde{Q}$, let (in the non-decreasing order of $|\tilde{Q}|$)

$$\Delta(\delta,\tilde{Q},\gamma) \leftarrow \Delta(\delta,\tilde{Q},\{\alpha\},\alpha) + \Delta_{\gamma} \text{ for any } \alpha \in \tilde{Q},$$

$$f^{\ell}(\delta,\tilde{Q},\gamma) \leftarrow \begin{cases} G_m \ (\Delta(1,Q,\delta)+\Delta(\delta,\tilde{Q},\gamma)-\Delta_{\delta}(=\Delta(1,Q \cup \{\delta\} \cup \tilde{Q}, \\ \quad \gamma)) \geq 0 \text{ and } G_m \leq 0) \\ \infty \ \text{(otherwise)}, \end{cases}$$

and

$$q(\delta,\tilde{Q},\gamma) \leftarrow (q(\delta,Q-\{\alpha^*\},\alpha^*),\gamma) \text{ if } f(\delta,\tilde{Q},\gamma) < \infty,$$

where

$$G_m = \min_{\alpha \in \tilde{Q}} \{c_{\alpha\gamma} + f^{\ell}(\delta,\tilde{Q}-\{\alpha\},\alpha)\}$$

$$= c_{\alpha^*\beta} + f^{\ell}(\delta, \tilde{Q}-\{\alpha^*\}, \alpha^*).$$

Substep 2c: For $\tilde{Q}$ and γ with $\tilde{Q} \cup \{\gamma\} = \bar{Q}$, let

$$f^{\ell}(\delta, \bar{Q}) \leftarrow G_m,$$

and

$$q(\delta, \bar{Q}) \leftarrow q(\delta, \bar{Q}-\{\gamma^*\}, \gamma^*) \text{ if } f^{\ell}(\delta, \bar{Q}) < \infty,$$

where

$$G_m = \min_{\gamma \in \bar{Q}} \{f^{\ell}(\delta, \bar{Q}-\{\gamma\}, \gamma)\}$$

$$= f^{\ell}(\delta, \bar{Q}-\{\gamma^*\}, \gamma^*).$$

Substep 3: Let

$$\pi^* \leftarrow \begin{cases} \pi^* & (f^{\ell}(1, Q^*, \delta^*) \geq d^*) \\ (q(1, Q^*, \delta^*),\ q(\delta^*, \bar{Q}^*)) & \text{(otherwise)}, \end{cases}$$

and

$$d^* \leftarrow \min[d^*,\ f^{\ell}(1, Q^*, \delta^*)],$$

where

$$f^{\ell}(1, Q^*, \delta^*) = \min[f^{\ell}(1, Q, \delta) \mid \delta \leq \ell,\ |Q| = \ell-2,\ Q \subset V-\{1, \delta\}].$$

Return to Step 2 of Phase I.

Theorem 4.2. The above algorithm terminates in a finite number of steps, and d^* upon termination is equal to the optimal value of $\bar{B}$.

Proof: The finiteness directly follows from the finite-

ness of V. To prove that the optimal value is obtained, first note that $f^{\ell}(1,Q,\delta)$ calculated in Step 1 of Phase II, has the interpretation as mentioned prior to the algorithm description. (This is a direct application of the principle of optimality used in dynamic programming. It is similar to the method discussed in Section 4.2.) $f^{\ell}(\delta,\bar{Q})$ calculated in Step 2 of Phase II may be interpreted as follows: $f^{\ell}(\delta,\bar{Q}) \leqq 0$ if there exists a subpath $q = (\delta_0=\delta, \delta_1, \cdots, \delta_{n-\ell})$, that starts from δ and passes through all the vertices in $\bar{Q}$ satisfying

$$\max_k \{ \sum_{i=1}^{k-1} c_{\delta_i \delta_{i+1}} \mid k=1,2,\cdots,n-\ell\} \leqq 0 \qquad (4.3)$$

$$\sum_{k \in Q \cup \{1,\delta\}} \Delta_k + \min_k \{ \sum_{i=1}^{k} \Delta_{\delta_i} \mid k=1,\cdots,n-\ell\} \leqq 0 \quad (4.4)$$

(Note that $\sum_{k \in Q \cup \{1,\delta\}} \Delta_k$ is the margin of the truck capacity at δ.) Thus, the subpath corresponding to $f^{\ell}(1,Q,\delta)$ such that its length assumes the maximum at δ (in this portion) can be completed by subpath q corresponding to $f^{\ell}(\delta,\bar{Q})$. The resulting path still assumes its maximum length at δ (by (4.3)) and satisfies the capacity constraint (by (4.4)).

Next, we show that an optimal path π^* (see Figure 4.1; π^* assumes its maximum at vertex δ) is in fact obtained by the above computation. For that, it is sufficient to prove that the π^*'s first portion p provides $f^{\ell}(1,Q,\delta)$ (i.e., $f^{\pi^*}_{\pi^*(\ell)} = f^{\ell}(1,Q,\delta)$) where $\ell = |Q|+2$. This is proved below.

Assume that $f^{\pi^*}_{\pi^*(\ell)} > f^{\ell}(1,Q,\delta)$, i.e., there exists a path $\hat{p}$ that starts from vertex 1, passes through the vertices in Q (in an order different from p) and reaches vertex δ, giving $f^{\ell}(1,Q,\delta)$ smaller than $f^{\pi^*}_{\pi^*(\ell)}$ (see Figure 4.1). Then, $\pi=(\hat{p},q)$ is also a path satisfying the capacity constraint and having its maximum at vertex $\delta \epsilon Q$. This implies that π is found (or a similar argument may again be applied to π) when $\bar{P}_{\hat{\ell}}$ is solved, where $\hat{\ell}+|\hat{Q}|+2 < \ell$. As a results, we have $d^* \leq f^{\pi}$ when $\bar{P}_{\ell}$ is solved. Thus, $f^{\ell}(1,Q,\delta)$ is set to ∞ in Substep 1b of Phase II, there-by contradicting the notion that $f^{\ell}(1,Q,\delta)$ $(Q \supset \hat{Q} \cup \{\delta\})$ is obtained corresponding to π. □

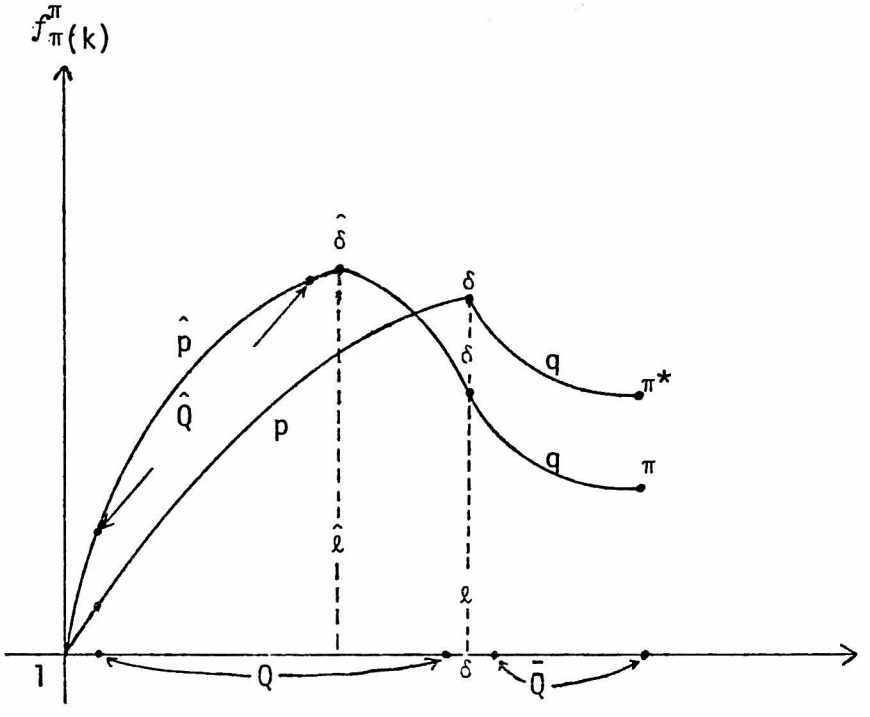


Figure 4.1. The relation between π and π^*.

4.6 Example

Consider the five-city problem given in Tables 4.1-4.3. The computation process is illustrated in Tables 4.4-4.8. An optimal route obtained is (1,5,4,2,3) and its value is 21. This route is shown in Figure 4.2.

Table 4.1. t_{ij} ($1 \le i \le n$, $1 \le j \le n$, $i \ne j$)

i \ j	1	2	3	4	5
1		13	6	9	3
2	7		2	4	1
3	6	8		3	2
4	2	3	7		1
5	9	2	2	3	

Table 4.2. t_i, r_i, w_i, Δ_i ($t_1=0$, $\Delta_i=r_i-w_i$, $\Delta=37$, $\Delta_1=\Delta-\sum_{i=2}^{5} r_i$)

i	t_i	w_i	r_i	Δ_i
1	0			7
2	12	10	13	3
3	9	9	3	-6
4	5	5	8	3
5	3	11	6	-5

Table 4.3. $C_{ij}=t_{ij}+t_j-t_i$
$(1\le i\le n,\ 1\le j\le n,\ i\neq j)$

i \ j	1	2	3	4	5
1		25	15	14	6
2	-5		-1	-3	-8
3	-3	11		-1	-4
4	-3	10	11		-1
5	6	11	8	5	

Table 4.5. Calculation of $f^{\ell}(\delta,\bar{Q})$ for $f^{2}(1,\phi,2)$.

	Step (2a)(2b)		
	$(\delta,\tilde{Q},\gamma)$	$f^{\ell}(\delta,\tilde{Q},\gamma)$	$q(\delta,\tilde{Q},\gamma)$
$\lvert\tilde{Q}\rvert=0$	$(2,\phi,3)$	-1	(3)
	$(2,\phi,4)$	-3	(4)
	Others	∞	—
$\lvert\tilde{Q}\rvert=1$	$(2,\{3\},4)$	-2	(3,4)
	$(2,\{5\},4)$	-3	(5,4)
	$(2,\{4\},5)$	-4	(4,5)
	Others	∞	—
$\lvert\tilde{Q}\rvert=2$	$(2,\{3,4\},5)$	-3	(3,4,5)
	Others	∞	—

Table 4.4. Computational process of the example in Section 4.6 (continued in next page).

$\bar{P}_\ell$	Step 1						
	Step 1a Step 1b				Step 1c		
	$(1,Q,\beta)$	$f^\ell(1,Q,\beta)$	$M(1,Q,\beta)$	$q(1,Q,\beta)$	$(1,Q,\delta)$	$f^\ell(1,Q,\delta)$	$q(1,Q,\delta)$
$\bar{P}_2$	—	—	—	—	$(1,\phi,2)$	25	$(1,2)$
$\bar{P}_3$	$(1,\phi,2)$	∞	—	—			
	$(1,\phi,3)$	15	15	$(1,3)$	Others	∞	—
	$(1,\phi,4)$	14	14	$(1,4)$	$\rightarrow(1,\{4\},2)$	24	$(1,4,2)$
	$(1,\phi,5)$	6	6	$(1,5)$	$\rightarrow(1,\{5\},2)$	17	$(1,5,2)$
$\bar{P}_4$	$(1,\phi,2)$	∞	—	—			
	$(1,\phi,3)$	15	15	$(1,3)$	Others	∞	—
	$(1,\phi,4)$	14	14	$(1,4)$			
	$(1,\phi,5)$	6	6	$(1,5)$			
	$(1,\{3\},4)$	14	15	$(1,3,4)$			
	$(1,\{4\},5)$	13	14	$(1,4,5)$	$\searrow(1,\{4,5\},2)$	21	$(1,5,4,2)$
	$(1,\{5\},4)$	11	11	$(1,5,4)$	$\nearrow$		
	$(1,\{5\},2)$	17	17	$(1,5,2)$			
	Others	∞	—	—			
$\bar{P}_5$	All	∞	—	—			

Table 4.4 (continued).

Step 2				Step 3		
Step 2a	Step 2c			π^*	d*	
Step 2b	$(\delta,\bar{Q})$	$f^{\ell}(\delta,\bar{Q})$	$q(\delta,\bar{Q})$			
(1,2) → Table 4.5	(2,{3,4,5})	-3	(3,4,5) →	(1,2,3,4,5)	25	← P_2
(1,4,2) → Table 4.6	(2,{3,5})	-5	(3,5) →	(1,4,2,3,5)	24	← P_3
(1,5,2) → Table 4.7	(2,{3,4})	∞	—			
(1,5,4,2) → Table 4.8	(2,{3})	-1	(3) →	(1,5,4,2,3)	21	← P_4

Table 4.6. Calculation of $f^{\ell}(\delta,\bar{Q})$ for $f^{3}(1,\{4\},2)$.

Step(2a)(2b)			
	$(\delta,\tilde{Q},\gamma)$	$f^{3}(\delta,\tilde{Q},\gamma)$	$q(\delta,\tilde{Q},\gamma)$
$\lvert\tilde{Q}\rvert=0$	$(2,\phi,3)$	-1	(3)
	$(2,\phi,5)$	-8	(5)
$\lvert\tilde{Q}\rvert=1$	$(2,\{3\},5)$	-5	(3,5)
	$(2,\{5\},3)$	0	(5,3)

Table 4.7. Calculation of $f^{\ell}(\delta,\bar{Q})$ for $f^{3}(1,\{5\},2)$.

Step(2a)(2b)		
$(\delta,\tilde{Q},\gamma)$	$f^{3}(\delta,\tilde{Q},\gamma)$	$q(\delta,\tilde{Q},\gamma)$
$(2,\phi,4)$	-3	(4)
Others	∞	—

Table 4.8. Calculation of $f^{\ell}(\delta,\bar{Q})$ for $f^{4}(1,\{4,5\},2)$

Step(2a)(2b)		
$(\delta,\tilde{Q},\gamma)$	$f^{4}(\delta,\tilde{Q},\gamma)$	$q(\delta,\tilde{Q},\gamma)$
$(2,\phi,3)$	-1	(3)

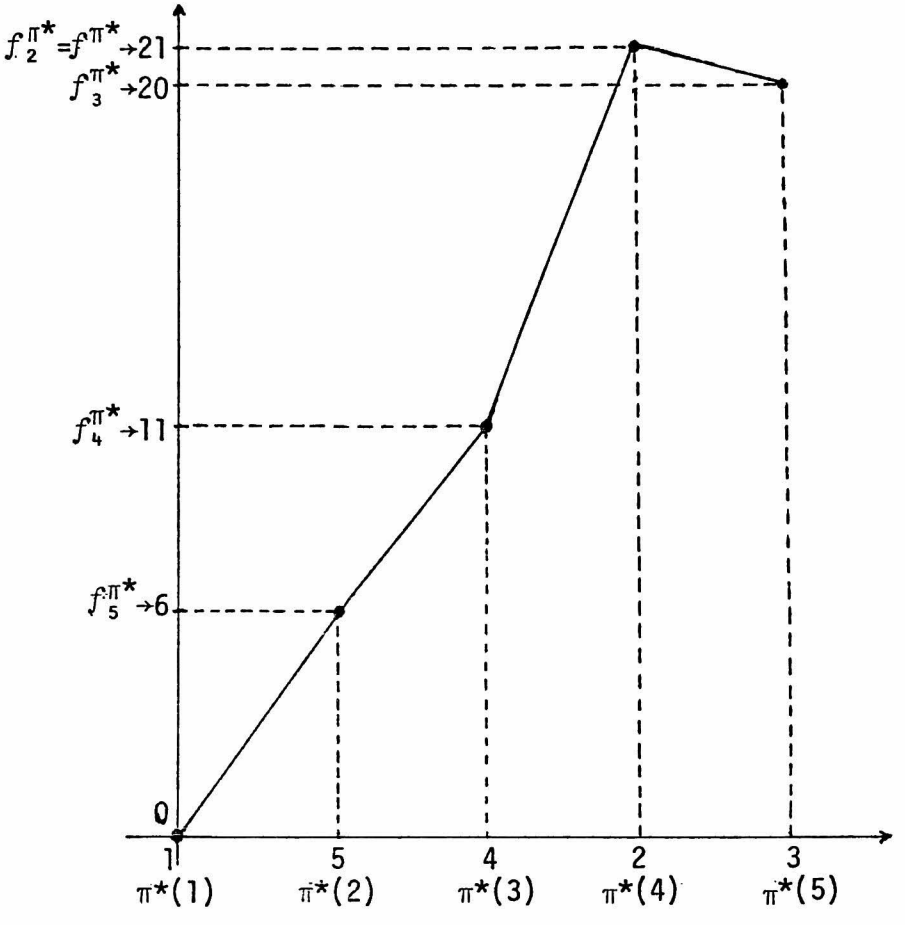


Figure 4.2. An optimal solution π^* of the example.
(an optimal route;1→5→4→2→3)

4.7 Conclusion

This chapter introduced the capacity constraint into the minimax type traveling salesman problem and the ordinary traveling salesman problem. It has been shown that the well-known dynamic programming approach for the traveling salesman problem by Held-Karp-Bellman, can be directly generalized to incorporate the capacity constraint. On the other hand, somewhat different decomposition scheme is required to solve the minimax type traveling salesman problem. The original problem is in this case decomposed into (n-1) subproblems according to the visiting order of the delivery truck. Then the solutions for subproblems obtained by the dynamic programming algorithm are combined into an optimal solution of the original problem in a manner similar to the algorithm of Chapter 3. Because of the existence of the capacity constraint, some feasible solutions of the original traveling salesman problems are infeasible in the problems of this chapter. It is expected that this kind of constrained traveling salesman problems may find a wide application area in practical problems.

To improve the algorithm discussed in this chapter, the incorporation of a lower bound for each subproblem P_ℓ into the algorithm may be helpful, because it may exclude in advance some impromising subproblems not containing any optimal solutions. In general, further investigation of techniques of branch-and-bound method would be worthwhile.

CHAPTER 5 A PRIMAL CUTTING PLANE ALGORITHMS FOR INTEGER FRACTIONAL PROGRAMMING PROBLEMS

5.1 Introduction

This chapter discusses an integer programming problem with a linear fractional objective function. Generally speaking, fractional programming problems reflect actual situations where optimization of certain ratio (e.g., system effectiveness per life cycle cost) is required. A marine transportation problem is such an example, in which one is interested in determining the best combination of cargoes to be loaded into a ship so that the ratio of the net profit to the total trip time is maximized. (For details, see [Pollack,E.G., G.N. Novaes and E.G.Frankel,'65] [Bitran,G.R. and A.G.Novaes, '73].)

Charnes and Cooper published the first paper on the fractional programming problem, which deals with a linear fractional programming problem. The idea of the solution procedure is to transform the problem into the equivalent linear programming problem. An optimal solution of the linear fractional programming problem is obtained by solving the equivalent linear programming problem. They also presented a duality theory for the linear fractional programming problem utilizing the duality of the linear programming problem. Short time ago, Shaible [Shaible, '76a] generalized their results into the concave fractional programming problem. Martos [Martos,'64] also proposed a simplex algorithm for the linear fractional

programming problem at almost the same time as Charnes and Cooper. For nonlinear fractional programming problems, R.Jagannathan [Jagannathan,'66] and Dinkelbach [Dinkelbach,'67] proposed algorithms based on parametric procedures. Recently, I.M.Stancu-Minasian [Stancu-Minasian,I.M.,'77] compiled a list of 321 articles of fractional programming. The bibliography covers the years 1960-1976 and it seems to include almost all papers on fractional programming.

As outlined above, research on fractional programming problems has mainly concentrated on continuous type problems. The integer fractional programming problem discussed in this chapter has a linear fractional objective function and the constraint that all variables are integer variables. This problem has so far attracted relatively little attention [Hammer and Redeanu,'68] [Robillard,'71] [Gruspan,'73]. We will add in this chapter a new algorithm which is an extention of Martos' method for the linear fractional programming problem with continuous variables [Martos,'64] to the integer problem. This extension is made possible by employing Young's simplified primal algorithm (SPA) to solve the pure integer programming problems, which arise when the original problem is solved in a parametric programming manner proposed by Jagannathan and Dinkelbach.

Section 5.2 gives general properties of fractional programming problems necessary to the discussion of Chapter 5-7. In Section 5.3, the problem is formulated and a subsidiary problem with a parameter is defined. Section

5.4 proposes an algorithm after the brief review of Young's SPA. Section 5.5 gives an illustrative example to show the entire scheme of the algorithm. Section 5.6 proves the validity and finiteness of the algorithm. In Section 5.7, some further considerations and conclusion are given. Finally, Appendix describes cut generation procedure by Young's SPA used as substeps in the algorithm.

5.2 General Properties of Fractional Programming Problems

This section gives general properties of fractional programming problems necessary to the discussion of this chapter and Chapters 6 and 7. Let us consider the following fractional programming problem P:

$$\text{P:} \quad \text{Maximize} \quad N(x)/D(x)$$

$$\text{subject to} \quad x \in S \subseteq R^n . \tag{5.1}$$

The following are assumed throughout Chapter 5-7 except. when specified otherwise.

(i) S is bounded and nonempty.

(ii) $D(x) > 0$ for all $x \in S$.

The next subsidiary problem $P(\xi)$ for $\xi \geq 0$ is associated with the problem P.

$$P(\xi): \quad \text{Maximize} \quad Z(x) \triangleq N(x) - \xi D(x)$$

$$\text{subject to} \quad x \in S .$$

The maximum value and the optimal solution of $P(\xi)$ are denoted by Z_ξ and x^ξ respectively. It is well known that P and $P(\xi)$ are closely related to each other as shown by the following Theorems 5.1-5.3. (Theorem 5.1 and Thorem 5.2 are due to [Jagannathan,'66] [Dinkelbach,'67]).

Theorem 5.1. Let ξ^* be the optimal value of P. Then it holds that

(i) $\xi < \xi^* \rightleftharpoons Z_\xi > 0.$

(ii) $\xi = \xi^* \rightleftharpoons Z_\xi = 0.$

(iii) $\xi > \xi^* \rightleftharpoons Z_\xi < 0.$

Theorem 5.2. Z_ξ is a strictly decreasing convex function of ξ.

By Thorem 5.1, the problem P reduces to finding the value of ξ such that $Z_\xi = 0$.

Theorem 5.3. For $x, \bar{x} \in S$ and $\xi = N(x)/D(x)$,

$$N(\bar{x}) - \xi D(\bar{x}) > 0 \rightleftharpoons N(\bar{x})/D(\bar{x}) > N(x)/D(x) .$$

Proof: $N(x) - \xi D(x) > 0 \rightleftarrows N(\bar{x}) - (N(x)/D(x))D(\bar{x}) > 0$

$\rightleftarrows N(\bar{x})/D(\bar{x}) - N(x)/D(x) > 0$ (since $D(\bar{x}) > 0$) □

The property of Theorem 5.3 plays an important role in the development of our algorithms in Chapters 5-7.

5.3 Integer Fractional Programming Problem

This chapter treats the following integer fractional programming problem P.

$$\text{P:}\quad \text{Maximize}\quad N(x)/D(x)=\Big(\sum_{j=1}^{n} c_j x_j + c_0\Big)\Big/\Big(\sum_{j=1}^{n} d_j x_j + d_0\Big)$$

subject to $x \in S$,

where

$$S=\{x=(x_1,\cdots,x_n) \mid \sum_{j=1}^{n} a_{ij} x_j \leqq b_i,\ i=1,\ldots,m,\ \text{and}$$

$$x_j \geqq 0,\ \text{integer},\ j=1,\ldots,n\}$$

and coefficients c_j, d_j, b_i, a_{ij} are all integers.
Obviously, results of Section 5.2 hold for this problem P.
Especially, subsidiary problem P(ξ) is defined as follows:

$$\text{P}(\xi)\text{:}\quad \text{Maximize}\quad N(x)-\xi D(x)=\sum_{j=1}^{n} (c_j-\xi d_j) x_j + c_0 - \xi d_0$$

subject to $x \in S$.

5.4 An Algorithm for the Integer Fractional Programming Problem

Before describing an algorithm for the integer fractional programming problem, we shall briefly review Young's SPA used to solve ordinary integer programming problems ([Young,'63 and '68] [Garfinkel and Nemhauser, '72] [Salkin,'75]). Young's SPA starts with the following integer programming problem as the initial tableau:

$$\begin{aligned} &\text{Maximize} \quad x_0 = c_0 + \sum_{j=1}^{n} (-c_j)(-x_j) \\ &\text{subject to} \quad x_{n+i} = b_i + \sum_{j=1}^{n} a_{ij}(-x_j) \ , \ i=1,\ldots,m, \\ &\qquad\qquad x_k \geq 0, \ k=1,2,\ldots,n+m, \end{aligned} \tag{5.2}$$

where the primal feasibility condition

$$b_i \geq 0, \quad i=1,2,\ldots,m$$

is assumed, and c_j, a_{ij}, b_i are all integer constants.

To deal with a tableau obtained after pivot operations, in general, let us assume that the following is the current tableau.

$$\begin{aligned} &\text{Maximize} \quad x_0 = \alpha_{00} + \sum_{j=1}^{n} \alpha_{0j}(-t_j) \\ &\text{subject to} \quad u_i = \alpha_{i0} + \sum_{j=1}^{n} \alpha_{ij}(-t_j) \ , \ i=1,\ldots,m', \end{aligned}$$

$$t_j \geqq 0 , \quad j=1,2,\ldots,n, \tag{5.3}$$

$$u_i \geqq 0 , \quad i=1,2,\ldots,m',$$

where the primal feasibility

$$\alpha_{i0} \geqq 0 , \quad i=1,2,\ldots,m'$$

is also assumed; u_i denotes the current basic variables and t_j nonbasic variables.

If this tableau satisfies the dual feasibility condition also, i.e.,

$$\alpha_{0j} \geqq 0, \quad j=1,\ldots,n,$$

then the current tableau provides an optimal solution and computation terminates. Otherwise, a cut is generated according to a certain rule, and a pivot operation is performed on this cut row. (m' in (5.3) includes the number of the generated cuts.) The resulting tableau satisfies one of the following three conditions:

Case (i): It is dual feasible. Then Young's SPA terminates.

Case (ii): It is not dual feasible, but will be able to satisfy $\alpha'_{00} > \alpha_{00}$ in the next tableau, where α'_{00} is the new coefficient α_{00} after the pivot operation is executed. (This case is called a *transition cycle.*) Then the same procedure is repeated by regarding the resulting tableau

as (5.2) until Case (i) or Case (iii) is reached. (If the tableau has L-row (introduced in Case (iii) below), it is deleted before the pivot operation.)

Case (iii): It is not dual feasible, and $\alpha'_{00} = \alpha_{00}$ will hold in the next tableau. (This case is called a *stationary cycle.*) A pivot element is determined by a rule based on a special row, called *L-row.* (if the current tableau does not have L-row, a pivot element is selected after generating L-row.) Then pivot operation is performed. After repeating this process finite times, it is known either Case (i) or Case (ii) is eventually reached.

By repeating the above cycles, Young's SPA guarantees that Case (i) is eventually reached and an optimal solution of (5.2) is reached. Now note that the problem P(ξ) is rewritten as follows:

$$
\begin{aligned}
\text{P}(\xi)\text{: Maximize } & x_0 = c_0 - \xi d_0 + \sum_{j=1}^{n} \{(-c_j) - \xi(-d_j)\}(-x_j) \\
\text{subject to } & z_1 = c_0 + \sum_{j=1}^{n} (-c_j)(-x_j) , \\
& z_2 = d_0 + \sum_{j=1}^{n} (-d_j)(-x_j) , \qquad (5.4) \\
& x_{n+i} = b_i + \sum_{j=1}^{n} a_{ij}(-x_j), \quad i=1,2,\ldots,m \\
& x_j \geq 0, \text{ integer, } j=1,2,\ldots,n+m.
\end{aligned}
$$

This differs from (5.2) only in that the objective function is parametrized by ξ, and rows z_1 and z_2 are

augmented. The primal feasibility

$$b_i \geq 0 \ , \ i=1,2,\ldots,m,$$

is also assumed. For a fixed ξ, our algorithm is exactly the same as Young's SPA. Two rows z_1 and z_2 correspoding to $N(x)$ and $D(x)$ respectively, are used to compute the new objective row x_0 when ξ is modified. To describe a general step of our algorithm, let the current tableau be as follows:

$$\text{Maximize} \quad x_0 = \alpha_{00}(\xi) + \sum_{j=1}^{n} \alpha_{0j}(\xi)(-t_j)$$

$$\text{subject to} \quad z_1 = \beta_{00} + \sum_{j=1}^{n} \beta_{0j}(-t_j)$$

$$z_2 = \gamma_{00} + \sum_{j=1}^{n} \gamma_{0j}(-t_j)$$

$$u_i = \alpha_{i0} + \sum_{j=1}^{n} \alpha_{ij}(-t_j) \ , \quad i=1,2,\ldots,m',$$

$$t_j \geq 0, \quad j=1,2,\ldots,n,$$

$$u_i \geq 0, \quad i=1,2,\ldots,m'.$$

This tableau satisfies the primal feasibility

$$\alpha_{i0} \geq 0, \quad i=1,2,\ldots,m'$$

and all coefficients α_{ij}, β_{0j}, γ_{0j} are integers. The objective row x_0 is related to the z_1 and z_2 rows in the

following way;

$$\alpha_{0j}(\xi) = \beta_{0j} - \xi\gamma_{0j} \quad , \quad j=1,2,\ldots,n. \qquad (5.5)$$

Our algorithm starts with the initial tableau (5.4) with

$$\xi = c_0/d_0$$

(this makes $\alpha_{00}(\xi) = 0$). If the initial tableau does not satisfy the dual feasibility condition, the above Young's SPA is applied until coefficient $\alpha_{00}(\xi)$ strictly increases (i.e., $\alpha_{00}(\xi) > 0$) or the dual feasibility condition (with $\alpha_{00}(\xi) = 0$) is satisfied. As noted above, Young's SPA always produces one of the two results in finite pivot operations. If a dual feasible tableau is obtained, computation terminates and the resulting tableau provides an optimal solution. If $\alpha_{00}(\xi) > 0^{+}$ is obtained, however, ξ is updated to $\xi = \beta_{00}/\gamma_{00}$ (this is justified by Theorem 5.3) so that $\alpha_{00}(\xi) = 0$ and the objective row x_0 is recalculated by (5.5). Then the above procedure is repeated.

As shown in Theorem 5.4, the entire computation eventually terminates and an optimal solution is obtained.

An Algorithm for the Integer Fractional Programming Problem

Step 1: (Initialize)†† Let $\xi \leftarrow N(x^I)/D(x^I)$ and $x_j^I \leftarrow 0$, $j=1,2,\ldots,n$.

Step 2: (Check the optimality)††† If

---

† †† ††† See next page footnotes.

$$\alpha_{0j}(\xi) \geq 0, \quad j=1,2,\ldots,n,$$

go to Step 4. otherwise, add a cut and apply a pivot operation according to the rule for the transition cycle of Young's SPA (see Appendix for details).

Step 3: If $\alpha_{00}(\xi) > 0$, let $\xi \leftarrow \beta_{00}/\gamma_{00}$ and update the x_0-row by (5.4). Return to Step 2. Otherwise return to Step 2 directly.

Step 4: (Terminate) Terminate. The current ξ is the optimal value of P and

$$u_i = \alpha_{i0}, \quad i=1,2,\ldots,m'$$

$$t_j = 0, \quad j=1,2,\ldots,n$$

is an optimal solution of P.

---

†$\alpha_{00}(\xi) > 0$ corresponds to $\alpha'_{00} > \alpha_{00}$ (the condition to enter the transition cycle).

††We assume that the tableau (5.4) obtained from the above ξ is primal feasible, i.e.,

$$b_i \geq 0, \quad i=1,2,\ldots,m.$$

It obviously satisfies $\alpha_{00}(\xi) = 0$. If (5.4) does not provide a primal feasible tableau, a primal feasible tableau has to be obtained by some means. This point is not discussed, since it is the same as the ordinary integer programming problem.

†††When Step 2 is entered, $\alpha_{00}(\xi) = 0$ is always satisfied.

5.5 Example

Consider the following problem P (see Figure 5.1.)

P: Maximize $$\frac{N(x)}{D(x)} = \frac{3 + 7x_1 + 9x_2}{2 + 3x_1 + 4x_2}$$

subject to $$x_3 = 6 + 2(-x_1) + 3(-x_2)$$

$$x_4 = 5 + 3(-x_1) + 2(-x_2)$$

$$x_1, x_2, x_3, x_4 \geq 0, \text{ integer}.$$

From the constraints we have

$$0 \leq x_1 \leq min\{ \left\lfloor \frac{6}{2} \right\rfloor , \left\lfloor \frac{5}{2} \right\rfloor \} = 1 \ (= U_1)$$

$$0 \leq x_2 \leq min\{ \left\lfloor \frac{6}{3} \right\rfloor , \left\lfloor \frac{5}{2} \right\rfloor \} = 2 \ (= U_2) \quad (5.6)$$

$$0 \leq x_3 \leq 6 \ (=U_3), \qquad 0 \leq x_4 \leq 5 \ (=U_4),$$

where $\lfloor x \rfloor$ denotes the integer part of x.

Step 1: Let $\xi \leftarrow N(x^I)/D(x^I)$ $(= c_0/d_0 = 3/2)$ and construct the initial tableau (Table 5.1) for

$$x^I = (x_1^I, x_2^I, x_3^I, x_4^I) = (0,0,6,5).$$

Step 2: Since $\alpha_{01}(\xi) = -5/2$, $\alpha_{02}(\xi) = -6/2$, the dual feasibility condition is not satisfied. We have $J_p = \{1,2\}$ according to *Substep* 1 of Young's SPA since $\theta_1 = min\{6/2, 5/3\}$

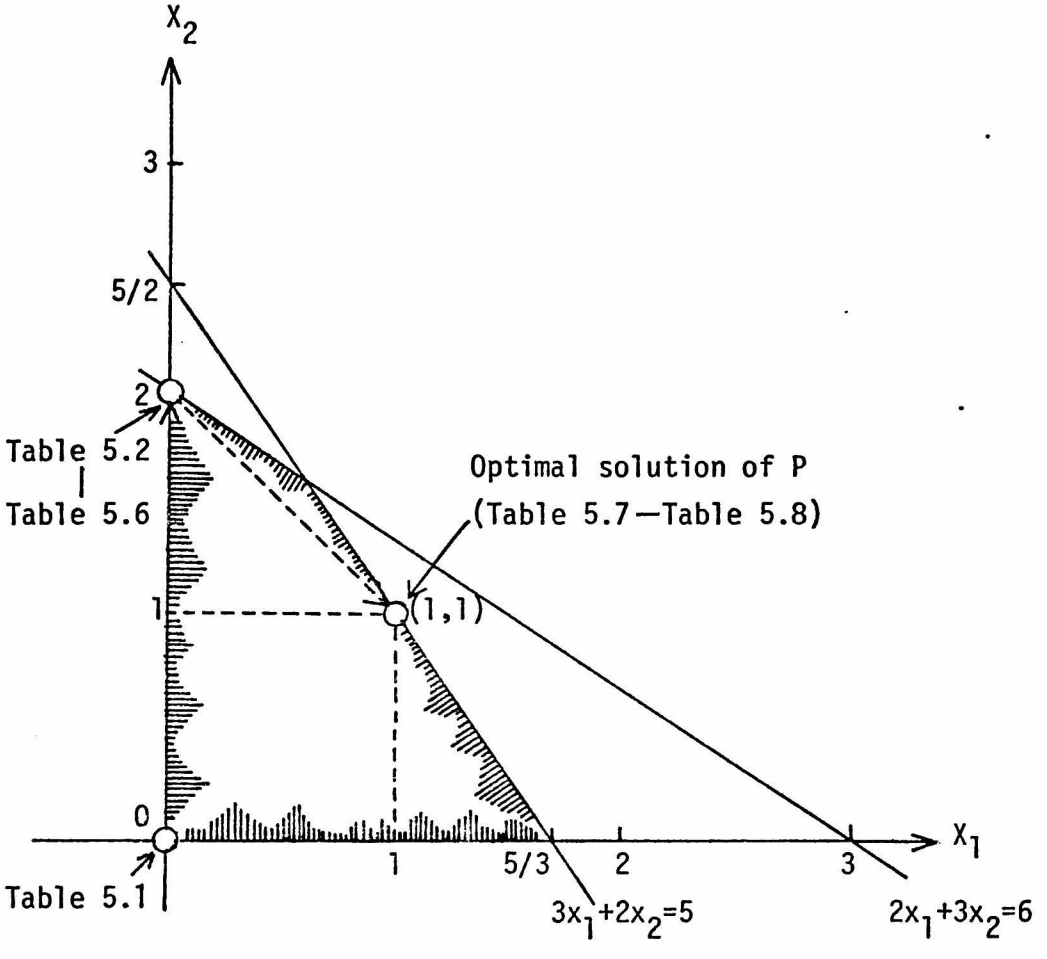


Figure 5.1. Illustration of computational process for the example in Section 5.5.

$=5/3 \geqq 1$, $\theta_2 = min\{6/3, 5/2\} = 2 \geqq 0$. In *Substep 2*, x_2 column is chosen as a pivot column though x_1 column is also possible. In *Substep 5*, cut

$$s_1 = 2 + 0(-x_1) + 1(-x_2) \tag{5.7}$$

is generated and added to the tableau (see s_1-row of Table 5.1) and execute a pivot operation (* denotes the pivot element). The resulting tableau is given in Table 5.2. Obviously

$$0 \leqq s_1 \leqq 2$$

follows from (5.6) and (5.7).

Step 3: $\alpha_{00}(\xi) = 6 > 0$. Thus let $\xi \leftarrow \beta_{00}/\gamma_{00} = 21/10$, and recalculate the x_0-row of Table 5.2 by using the new ξ. Table 5.3 shows the resulting tableau. Return to Step 2.

Step 2: $\alpha_{01}(\xi) = -7/10 < 0$; the dual feasibility is not satisfied. Since $J_p = \phi$ in this case and the tableau does not have L-row, L-row

$$x_L = 3 + 1(-x_1) + 1(-s_1)$$

is added in *Substep 3* (see x_L-row of Table 5.3; $\alpha_{L0} = 3$ $(=1+2)$ since $0 \leqq x_1 \leqq 1$ and $0 \leqq s_1 \leqq 2$). According to the rule for the stationary cycle of Young's SPA, x_1-column is selected as the pivot column in *Substep 4*. Then cut row is added to the tableau (see s_2-row of Table 5.3).

Table 5.1. First Tableau.

	1	$-x_1$	$-x_2$
$x_0 =$	0	-5/2	-6/2
$z_1 =$	3	-7	-9
$z_2 =$	2	-3	-4
$x_3 =$	6	2	3
$x_4 =$	5	3	2
$s_1 =$	2	0	1*

Table 5.2. Second Tableau.

	1	$-x_1$	$-s_1$
$x_0 =$	6	-5/2	6/2
$z_1 =$	21	-7	9
$z_2 =$	10	-3	4
$x_3 =$	0	2	-3
$x_4 =$	1	3	-2
$x_2 =$	2	0	1

Table 5.3. Third Tableau.

	1	$-x_1$	$-s_1$
$x_0 =$	0	-7/10	6/10
$z_1 =$	21	-7	9
$z_2 =$	10	-3	4
$x_3 =$	0	2	-3
$x_4 =$	1	3	-2
$x_2 =$	2	0	1
$x_L =$	3	1	1
$s_2 =$	0	1*	-2

Table 5.4. Fourth Tableau.

	1	$-s_2$	$-s_1$
$x_0 =$	0	7/10	-8/10
$z_1 =$	21	7	-5
$z_2 =$	10	3	-2
$x_3 =$	0	-2	1*
$x_4 =$	1	-3	4
$x_2 =$	2	0	1
$x_L =$	3	-1	3
$x_1 =$	0	1	-2

$$0 \leqq s_2 \leqq 4$$

follows from s_2-row and $0 \leqq s_1 \leqq 2$. The resulting tableau after a pivot operation is given in Table 5.4.

Step 3: $\alpha_{00}(\xi) = 0$. Return to Step 2.

Step 2: $\alpha_{00}(\xi) = -8/10 < 0$; the dual feasibility is not satisfied. Although $J_p = \phi$, *Substep 3* is skipped since Table 5.4 has L-row. In *Substep 4* and *Substep 5*, a cut is generated. In this case, however, the generated cut is same as x_3-row, and x_3-row is used as the pivot row. After a pivot operation, Table 5.5 is obtained.

Step 3: $\alpha_{00}(\xi) = 0$. Return to Step 2.

Step 2: $\alpha_{00}(\xi) = -9/10 < 0$. According to *Substep 4* and *Substep 5*, add a cut

$$s_3 = 0 + 1(-s_2) + 1(-x_3)$$

to the tableau, and execute a pivot operation, to obtain Table 5.6.

Step 3: $\alpha_{00}(\xi) = 0$. Return to Step 2.

Step 2: $\alpha_{02}(\xi) = -1/10 < 0$. Since $J_p = \{2\}$, in this case, a cut is generated according to *Substeps 2* and *5*. (Note that L-row is deleted). The generated cut is same as x_4-row. After a pivot operation, Table 5.7 is obtained.

Step 3: $\alpha_{00}(\xi) = 1/10 > 0$. Then let $\xi \leftarrow \beta_{00}/\gamma_{00} = 19/9$ and recalculate x_0-row. Table 5.8 is obtained.

Step 2: $\alpha_{01}(\xi) = 9/3 > 0$, $\alpha_{02}(\xi) = 1/9 > 0$; the dual feasibility is satisfied. Go to Step 4.

Step 4: Terminate. An optimal solution x^0 is given by

$$x_1^0 = x_2^0 = x_3^0 = 1, \quad x_4^0 = 0 \qquad N(x^0)/D(x^0) = \xi = 19/9.$$

Table 5.5. Fifth Tableau.

	1	$-s_2$	$-x_3$
x_0 =	0	-9/10	8/10
z_1 =	21	-3	5
z_2 =	10	-1	2
x_4 =	1	5	4
x_2 =	2	2	-1
x_1 =	0	-3	2
x_L =	3	5	-3
s_3 =	0	1*	-1

Table 5.6. Sixth Tableau.

	1	$-s_3$	$-x_3$
x_0 =	0	9/10	-1/10
z_1 =	21	3	2
z_2 =	10	1	1
x_4 =	1	-5	1*
x_2 =	2	-2	1
x_1 =	0	3	-1
x_L =	3	-5	2

Table 5.7. Seventh Tableau.

	1	$-s_3$	$-x_4$
x_0 =	1/10	4/10	1/10
z_1 =	19	13	-2
z_2 =	9	6	-1
x_3 =	1	-5	1
x_2 =	1	3	-1
x_1 =	1	-2	1

Table 5.8. Eighth Tableau.

	1	$-s_3$	$-x_4$
x_0 =	0	3/9	1/9
z_1 =	19	13	-2
z_2 =	9	6	-1
x_3 =	1	-5	1
x_2 =	1	3	-1
x_1 =	1	-2	1

5.6 Proof of Finiteness and Validity

Theorem 5.4 (Finiteness and Validity). The procedure given in Section 5.4 terminates in a finite number of iterations and, upon termination, produces an optimal solution of P.

Proof: (Finiteness): (a) From Assumption 2 of Section 5.2, the number of feasible solution in S is finite.
(b) Each P(ξ) has the same feasible region S, and each tableau represents a feasible solution since the primal feasibility is assumed. When ξ is updated in Step 3 or Step 5, the new ξ satisfies,

$$\xi = \frac{\beta_{00}}{\gamma_{00}} = \frac{N(\bar{x})}{D(\bar{x})} ,$$

where $\bar{x}$ is the feasible solution represented by the tableau. By Theorem 5.3, this new ξ is strictly greater than the old ξ.
(c) It is known that, for each ξ, condition $\alpha_{00}(\xi) > 0$ (then ξ is updated) or a dual feasibility (termination) is satisfied after a finite number of pivot operations. (This property was first proved by Young [Young,'63 and '68] under the assumption that all coefficients in a tableau are integers. H.Salkin, P.Schroff and S.Mehta [Salkin,et.al.,'74] generalized this property to the case in which each α_{ij} is rational. Note that $\alpha_{0j}(\xi)$ is rational in our algorithm.) (a)(b)(c) together prove the finiteness.

(Validity): When Step 6 is reached, the tableau satisfies

both primal and dual feasibility conditions, and hence it gives an optimal solution of $P(\xi)$ for the current ξ. Since the tableau also satisfies $\alpha_{00}(\xi) = 0$, this solution is an optimal solution of P by Theorem 5.1. □

5.7 Conclusion

This chapter has shown that the primal cutting plane algorithm for the ordinary integer programming problem can be easily extended to handle the integer fractional programming problem. The proposed algorithm uses Young's SPA as a subalgorithm and continue to generate some tableaux by pivot operations until some termination condition is met. In each stationary cycle, L-row is adjoined to the current tableau, until $\alpha_{00}(\xi)>0$ or the dual feasibility condition is satisfied. If the dual feasibility condition is satisfied, computation terminates and the optimal solution is given by the final tableau. If the condition $\alpha_{00}(\xi)>0$ results, the tableau should be updated so that $\alpha_{00}(\xi)=0$ holds by changing the value of parameter ξ. Then the above procedure is repeated. The validity and finiteness of the algorithm were proved in Theorem 5.4.

In view of the results obtained in this chapter, it appears possible to modify other primal algorithms such as Glover's simplified primal algorithm [Glover,'68] which seems to be better than Young's SPA according to the computational experiment [Salkin,'72]. However, a straightforward application of Glover's SPA seems to cause a difficulty because the required property of the reference equation is no longer preserved when parameter ξ is updated, where the reference equation in Glover's SPA plays a role similar to the L-row in Young's SPA. Thus it would be a subject for further research to find a generation method of the reference equation when ξ is updated.

Appendix Description of Cut Generation Procedure by Young's SPA ([Young,'68])

The generation of a cut and the selection of pivot element in Young's SPA are done according to the following rule. α_j (j=1,2,...,n) is used to denote the j-th column of a tableau. J_p computed during computation determines whether the present cycle is transitive or stationary; $J_p = \phi$ implies that $\alpha'_{00} > \alpha_{00}$ holds in the next tableau (i.e., it is a transitive cycle), while $J_p = \phi$ implies that $\alpha'_{00} = \alpha_{00}$ holds in the next tableau (i.e., a stationary cycle).

Substep 1: (This substep is entered from Step 2 of the main algorithm in Section 5.4): Calculate $\theta_j = min\{\alpha_{i0} / \alpha_{ij} | \alpha_{ij} > 0$, i does not correspond to x_0-row, z_1-row, z_2-row or L-row$\}$ for each j satisfying $\alpha_{0j}(\xi) < 0$. $J_p \leftarrow \{j | \theta_j \geq 1\}$. Go to Substep 2 if $J_p \neq \phi$, and go to Substep 3 if $J_p = \phi$.

Substep 2: If the current tableau has L-row, delete it from the tableau. Select any column α_{j_0}, $j_0 \in J_p$, As the pivot column. Go to Substep 5.

Substep 3: If the current tableau has L-row, go to Substep 4. Otherwise add the following L-row to the tableau, and go to Substep 4.

$$x_L = \alpha_{L0} + \sum_{j=1}^{n} (-t_j) ,$$

where

$$\alpha_{Lo} = \sum_{j=1}^{n} U_j$$

and U_j is an upper bound of t_j, i.e., $0 \leq t_j \leq U_j$ for any $x \in S$. (The example in Section 5.5 include a method for obtaining U_j.)

Substep 4: Let α_{Lj} (j=1,2,...,n) denote the elements in the L-row. For each α_j, j=1,2,...,n, that has $\alpha_{Lj} > 0$, calculate column

$$R_j = (\alpha_{0j}/\alpha_{Lj},\ \alpha_{1j}/\alpha_{Lj},\cdots,\ \alpha_{m'j}/\alpha_{Lj})' ,$$

where ' denotes transpose, and select the lexicographically smallest column among them as the pivot column α_{j_0}. Go to Substep 5.

Substep 5: Calculate i_0 such that

$$\alpha_{i_0 0}/\alpha_{i_0 j_0} = min\{\alpha_{i0}/\alpha_{ij_0} \mid \alpha_{ij_0} > 0,\ i=1,2,...,n\},$$

and generate the following Gomory cut to adjoin the tableau.

$$s = \left\lfloor \alpha_{i_0 0}/\upsilon \right\rfloor + \sum_{j=1}^{n} \left\lfloor \alpha_{i_0 j}/\upsilon \right\rfloor (-t_j) ,$$

where $\upsilon = \alpha_{i_0 j_0}$ (> 0) and $\lfloor y \rfloor$ denotes the integer part of y. Execute a pivot operation on the pivot element $\lfloor \alpha_{i_0 j_0}/\upsilon \rfloor$ (=1) of the s-row. (Then return to Step 3 of the main algorithm in Section 5.4).

CHAPTER 6 FRACTIONAL KNAPSACK PROBLEMS

6.1 Introduction

This chapter discusses fractional knapsack problem which is an integer programming problem to maximize a fractional objective function under the constraint given by one linear inequality.

As noted in Chapter 5, studies on fractional programming problems have been mostly concentrated on those with continuous variables, e.g., [Charnes and Cooper,'62] [Martos,'64] [Jagannathan,'66] [Dinkelbach,'67] [Elmagraby and Arisawa,'72] [Bitran and Novaes,'73]. Some extensions to the case of the integer fractional programming problems are known, e.g., [Hammer and Randeanu,'68] [Robillard,'71] [Gruspan,'73].

This chapter proposes an algorithm for solving the fractional knapsack problem, which is a modification of Dinkelbach method [Dinkelbach,'67] developed for rather general nonlinear fractional programming problems. The algorithm iteratively generates subsidiary problems with a parameter. Subsidiary problems are ordinary knapsack problems, and can be solved by any existing algorithm. ([Gilmore and Gomory,'66] [Garfinkel and Nemhauser,'72]). However, the generated subsidiary problems are usually not solved exactly, but good feasible solutions are computed by a heuristic, called greedy algorithm. If the computed feasible solution satisfies a certain condition, the parameter in our algorithm can be updated, and the next subsidiary problem is generated. If we follow

Dinkelbach method directly, each subsidiary problem is completely solved and the parameter value is updated based on the optimal solution of each subsidiary problem. It should be noted that good feasible solutions are easily obtainable for the ordinary knapsack problem [Hu,T.C. and M.L.Lenard,'73] [Magazine,M.J.,et.al.,'75] [Sahni,S.,'75] and also for the fractional knapsack problem as will be shown in Section 6.4. It is expected that the computational efficiency is greatly enchanced by this modification. To support this, the following theorem is proved: The number of the ordinary knapsack problems to be solved completely by the modified algorithm is not more than that due to Dinkelbach.

Following a precise definition in Section 6.2, various properties of optimal solutions useful in order to reduce the problem size (i.e., eliminate nonessential variables) are derived in Section 6.3. Section 6.4 investigates how optimal solutions behave as the right hand side b of the constraint inequality increases. In particular, it is shown that an optimal solution is easily obtained if b exceeds a certain constant. Section 6.5 introduces subsidiary problem with a parameter and investigates properties of this subsidiary problem. Based on the results shown in Section 6.2-6.5, a modified algorithm of Dinkelbach [Dinkelbach,'67] is proposed in Section 6.6. Section 6.6 proves that the efficiency of the modified algorithm is always not worse than that of Dinkelbach's, if measured by the number of ordinary knapsack problems to be solved completely until

termination. An example in Section 6.7 illustrates how the modified algorithm works. In Section 6.8, an upper bound for the number of knapsack problems to be solved in the modified and direct algorithms, is derived. Finally, Section 6.9 summarizes the results in this chapter.

6.2 Problem Statement

Consider the following fractional knapsack problem K.

$$\text{K: Maximize } K(x) \triangleq N(x)/D(x) \triangleq (c_0 + \sum_{k=1}^{n} c_k x_k)/(d_0 + \sum_{k=1}^{n} d_k x_k)$$

$$\text{subject to } \sum_{k=1}^{n} a_k x_k \leq b$$

$$x_k \text{ ; nonnegative integer, } k=1,2,\ldots,n,$$

where

$$c_k, d_k \text{; positive integers, } k=0,1,\ldots,n \tag{6.1}$$

$$c_1/d_1 \geq c_2/d_2 \geq \cdots \geq c_n/d_n > c_0/d_0 \tag{6.2}$$

$$0 \leq a_k \leq b, \quad a_k \text{; integer, } k=1,2,\ldots,n. \tag{6.3}$$

Remark 6.1. As is assumed in Section 5.2, it is natural to assume that $D(x) > 0$ for each feasible solution x (i.e., nonnegative integers $x_1, x_2, \cdots, x_n$ satisfying $\sum_{k=1}^{n} a_k x_k \leq b$). To guarantee $D(x) > 0$, assume here that $d_0, d_1, \cdots, d_n > 0$. Then, the assumption $c_0, c_1, \cdots, c_n > 0$ made in (6.1) does not lose any generality, since $x_j = 0$ can be assumed in an optimal solution for all j satisfying $c_j \leq 0$. To satisfy (6.2), variables may be arranged. $c_n/d_n > c_0/d_0$ can also be assumed since otherwise there exists an optimal solution with $x_\ell = 0$ for all ℓ satisfying $c_0/d_0 \geq c_\ell/d_\ell$, as obvious from Theorem 6.2 shown later (consider a feasible solution $\hat{x} = (0,0,\ldots,0)$ in Theorem 6.2).

6.3 Some Properties of Optimal Solutions

First three lemmas are given for the subsequent discussion. Some properties of optimal solutions of K are then described. Proofs of Lemmas 6.1 and 6.3 are easy and ommited.

Lemma 6.1. If $p, q, r, s > 0$ and $p/q \leqq r/s$, then

$$p/q \leqq (\lambda p + \mu r)/(\lambda q + \mu s) \leqq r/s,$$

where $\lambda, \mu \geqq 0$ and $\lambda + \mu > 0$. The left (res. right) equality holds if and only if $\mu = 0$ or $p/q = r/s$ (res. $\lambda = 0$ or $p/q = r/s$).

$$(\lambda_1 p + \mu_1 r)/(\lambda_1 q + \mu_1 s) \leqq (\lambda_2 p + \mu_2 r)/(\lambda_2 q + \mu_2 s)$$

holds if $\lambda_2 \mu_1 \leqq \lambda_1 \mu_2$ and $\lambda_1, \mu_1, \lambda_2, \mu_2 \geqq 0$ and $\lambda_1 + \mu_1 > 0$

$\lambda_2 + \mu_2 > 0$.

Lemma 6.2. If $p, q, r, s, t, u > 0$, $p/q < r/s \leqq t/u$ and $(p + t)/(q + u) \leqq (p + r)/(q + s)$, then

$$t \leqq r$$

holds.

Proof: Assume $t > r$. Then

$$(p + r)/(q + s) < (p + (t/r)r)/(q + (t/r)s) \text{ (by Lemma 6.1)}$$

$= (p + t)/(q + (ts/r)) \leqq (p + t)/(q + u)$ (by $r/s \leqq t/u$).

This contradicts the assumption. □

Lemma 6.3. If $p, q, r, s > 0$, $p/q \leqq r/s$ and $q < s$, then

$$r/s \leqq (r-p)/(s-q).$$

(Equality holds if and only if $p/q = r/s$.)

Corollary 6.1. Any optimal solution $x^* = (x_1^*, x_2^*, \cdots, x_n^*)$ of K satisfies

$$b \geqq \sum_{k=1}^{n} a_k x_k > b - a_1.$$

Proof: Since $c_0/d_0 < c_k/d_k \leqq c_1/d_1$, k=1,2,...,n, repeated applications of Lemma 6.1 show that

$$(c_0 + \sum_{k=1}^{n} c_k x_k^*)/(d_0 + \sum_{k=1}^{n} d_k x_k^*) < c_1/d_1.$$

If $\sum_{k=1}^{n} a_k x_k^* \leqq b - a_1$, then a new solution $\bar{x} = (x_1^* + 1, x_2^*, \cdots, x_n^*)$ is also feasible and satisfies

$$K(\bar{x}) = \frac{c_0 + \sum_{k=1}^{n} c_k \bar{x}_k}{d_0 + \sum_{k=1}^{n} d_k \bar{x}_k} = \frac{c_0 + \sum_{k=1}^{n} c_k x_k^* + c_1}{d_0 + \sum_{k=1}^{n} d_k x_k^* + d_1}$$

$$> \frac{c_0 + \sum_{k=1}^{n} c_k x_k^*}{d_0 + \sum_{k=1}^{n} d_k x_k^*} = K(x^*)$$

by Lemma 6.1. This is a contradiction to optimality of x^*. □

Theorem 6.1. If there exists integers $p_i, p_j > 0$ for $i < j$ such that

$$p_j a_j \geq p_i a_i \quad \text{and} \quad p_i c_i \geq p_j c_j ,$$

then there exists an optimal solution $x^* = (x_1^*, x_2^*, \cdots, x_n^*)$ satisfying

$$0 \leq x_j^* < p_j .$$

Proof: Assume $x_j^* \geq p_j$. Two cases are then possible.

Case (i): $(c_0 + \sum_{k=1}^{n} c_k x_k^*)/(d_0 + \sum_{k=1}^{n} d_k x_k^*) \geq c_j/d_j .$ (6.4)

Then $\bar{x} = (x_1^*, \cdots, x_{j-1}^*, 0, x_{j+1}^*, \cdots, x_n^*)$ satisfies

$$\sum_{k=1}^{n} a_k \bar{x}_k = \sum_{k=1}^{n} a_k x_k^* - a_j x_j^* < \sum_{k=1}^{n} a_k x_k^* \leq b$$

and

$$K(\bar{x}) = \frac{c_0 + \sum_{k=1}^{n} c_k \bar{x}_k}{d_0 + \sum_{k=1}^{n} d_k \bar{x}_k} = \frac{c_0 + \sum_{k=1}^{n} c_k x_k^*}{d_0 + \sum_{k=1}^{n} d_k x_k^*} = K(x^*)$$

by (6.4) and Lemma 6.3. Thus $\bar{x}$ is also optimal.

Case (ii): $(c_0 + \sum_{k=1}^{n} c_k x_k^*)/(d_0 + \sum_{k=1}^{n} d_k x_k^*) < c_j/d_j$. (6.5)

Then $\bar{x}=(x_1^*,\cdots,x_{i-1}^*,\overset{i}{\vee}{x_i^*+p_i},x_{i+1}^*,\cdots,x_{j-1}^*,\overset{j}{\vee}{x_j^*-p_j},x_{j+1}^*,\cdots,x_n^*)$ satisfies

$$\sum_{k=1}^{n} c_k \bar{x}_k = \sum_{k=1}^{n} c_k x_k^* + p_i a_i - p_j a_j \leq \sum_{k=1}^{n} a_k x_k^* \leq b$$

and

$$K(\bar{x}) = \frac{c_0 + \sum_{k=1}^{n} c_k \bar{x}_k}{d_0 + \sum_{k=1}^{n} d_k \bar{x}_k} = \frac{c_0 + \sum_{k=1}^{n} c_k x_k^* + c_i p_i - c_j p_j}{d_0 + \sum_{k=1}^{n} d_k x_k^* + d_i p_i - d_j p_j}.$$

If $d_i p_i - d_j p_j \leq 0$, then clearly

$$K(\bar{x}) = \frac{c_0 + \sum_{k=1}^{n} c_k \bar{x}_k}{d_0 + \sum_{k=1}^{n} d_k x_k} \geq \frac{c_0 + \sum_{k=1}^{n} c_k x_k^*}{d_0 + \sum_{k=1}^{n} d_k x_k^*} = K(x^*)$$

since $c_i p_i - c_j p_j \geq 0$. On the other hand, if $d_i p_i - d_j p_j > 0$, then

$$(c_i p_i - c_j p_j)/(d_i p_i - d_j p_j) \geq c_i/d_i$$

$$\left(\geqq c_j/d_j > (c_0 + \sum_{k=1}^{n} c_k x_k^*)/(d_0 + \sum_{k=1}^{n} d_k x_k^*)\right)$$

by Lemma 6.3 and $i < j$ (i.e., $c_i/d_i \geqq c_j/d_j$). Thus by Lemma 6.1,

$$K(\bar{x}) = \frac{c_0 + \sum_{k=1}^{n} c_k \bar{x}_k}{d_0 + \sum_{k=1}^{n} d_k \bar{x}_k} \geqq \frac{c_0 + \sum_{k=1}^{n} c_k x_k^*}{d_0 + \sum_{k=1}^{n} d_k x_k^*} = K(x^*) \quad .$$

Again $\bar{x}$ is an optimal solution of K. This procedure may be repeated until $x_j < p_j$ is attained. □

Corollary 6.2. If there exist integers p_i, $p_j > 0$ for $i < j$ such that

$$p_j a_j \geqq p_i a_i \text{ and } p_i d_i \geqq p_j d_j \ ,$$

then there exists an optimal solution $x^* = (x_1^*, \cdots, x_n^*)$ satisfying $0 \leqq x_j^* < p_j$.

Proof: From $c_i/d_i \geqq c_j/d_j$, $p_i d_i - p_j d_j \geqq 0$ implies $p_i c_i - p_j c_j \geqq 0$. Then apply Theorem 6.1. □

Remark 6.2. As special cases of Theorem 6.1, the following properties may be easily shown. These may be included in the algorithm of Section 6.6 since their condition can be easily checked.

(i) If $a_i \leqq a_j$ and $c_i \geqq c_j$ (or $d_i \geqq d_j$) hold for $i < j$, then $p_i = p_j = 1$ satisfy Theorem 6.1. Consequently $x_j^* = 0$ can be assumed in an optimal solution.

(ii) If $\lfloor a_i/a_j \rfloor c_i \geq c_j$ holds for $i < j$, then $p_i = \lfloor a_i/a_j \rfloor$ and $p_j = 1$ satisfy Theorem 6.1 where $\lfloor y \rfloor$ means the integer part of y. Thus $x_j^* = 0$ may again be assumed in an optimal solution.

Theorem 6.2. If there exists a feasible solution $x=(\hat{x}_1, \hat{x}_2, \cdots, \hat{x}_n)$ of K such that $K(\hat{x}) \geq c_j/d_j$, then there exists an optimal solution $x^* = (x_1^*, x_2, \cdots, x_n^*)$ satisfying $x_\ell^* = 0$, $\ell=j, j+1, \ldots, n$.

Proof: Let x^* be an optimal solution of K. Then

$$K(x^*) \geq K(\hat{x}) \geq c_\ell/d_\ell \quad , \quad \ell=j,\ldots,n.$$

Then the repeated applications of case (i) in the proof of Theorem 6.1 show that $\bar{x} = (x_1^*, \cdots, x_{j-1}^*, \underbrace{0, 0, \cdots, 0}_{n-j+1})$ is also optimal. □

6.4 Dependence of Optimal Solutions on b

This section describes how optimal solutions of K change when b (right hand side of the constraint) is increased. Assume that

$$c_1/d_1 = c_2/d_2 = \cdots = c_s/d_s \geq c_{s+1}/d_{s+1} \geq \cdots \geq c_n/d_n > c_0/d_0 \quad (6.6)$$

holds in (6.2). The following discussion in particular shows that K is reduced to an ordinary knapsack problem with s variables if b exceeds a certain bound determined from the problem coefficients; if s=1, the following solution $\hat{x}$ is always optimal:

$$\hat{x} = (\lfloor b/a_1 \rfloor, 0, \cdots, 0) \quad (6.7)$$

Now let

$$b_j = (a_1-1) + a_1(c_jd_0 - c_0d_j)/(c_1d_j - c_jd_1), \quad j=s+1,\ldots,n. \quad (6.8)$$

Obviously $b_{s+1} \geq b_{s+2} \geq \cdots \geq b_n$ holds since

$$b_i - b_j = \frac{\{((c_1/d_1)-(c_j/d_j))((c_i/d_i)-(c_0/d_0))-((c_1/d_1)-(c_i/d_i))((c_j/d_j)-(c_0/d_0))\}}{(c_1d_i-c_id_1)(c_1d_j-c_jd_1)} \times a_1d_0d_1d_id_j \geq 0 \text{ for } j > i \ (\geq s+1).$$

Theorem 6.3. If $b \geqq b_j$ $(j \geqq s+1)$, there exists an optimal solution x^* of K satisfying $x^*_\ell = 0$, $\ell=j, j+1,\ldots,n$.

Proof: Assume $b \geqq b_j$. Then $\hat{x}$ given in (6.7) is feasible and satisfies

$$K(\hat{x}) = \frac{c_0 + \lfloor b/a_1 \rfloor c_1}{d_0 + \lfloor b/a_1 \rfloor d_1}$$

$$\geqq \frac{c_0 + c_1(c_j d_0 - c_0 d_j)/(c_1 d_j - c_j d_1)}{d_0 + d_1(c_j d_0 - c_0 d_j)/(c_1 d_j - c_j d_1)} = c_j/d_j$$

by Lemma 6.1 since

$$b_j - (a_1(c_j d_0 - c_0 d_j)/(c_1 d_j - c_j d_1)) = a_1 - 1 \geqq b - \lfloor b/a_1 \rfloor$$

implies that

$$\lfloor b/a_1 \rfloor \geqq ((b - b_j)/a_1) + ((c_j d_0 - c_0 d_j)/(c_1 d_j - c_j d_1))$$

$$\geqq (c_j d_0 - c_0 d_j)/(c_1 d_j - c_j d_1) .$$

Thus by Theorem 6.2, there exists an optimal solution x^* satisfying $x^*_\ell = 0$, $\ell=j,\ldots,n$. □

Theorem 6.4. (i) If $b > b_{s+1}$ (s is defined in (6.6)), then

$$x^* = (\bar{x}_1, \bar{x}_2, \cdots, \bar{x}_s, 0, \cdots, 0)$$

is an optimal solution of K, where $\bar{x} = (\bar{x}_1, \bar{x}_2, \cdots, \bar{x}_s)$ is an optimal solution of the following knapsack problem:

$$\text{Maximize} \sum_{j=1}^{s} c_j x_j$$

$$\text{subject to} \sum_{j=1}^{s} a_j x_j \leq b, \tag{6.9}$$

$$x_j; \text{ nonnegative integer, } j=1,\ldots,s.$$

(ii) If s=1 and $b \geq b_2$, then $\hat{x}$ given in (6.7) is an optimal solution of K.

Proof: (i) By Theorem 6.3, $x_{s+1}=x_{s+2}=\cdots=x_n=0$ can be assumed in an optimal solution of K. Then K is reduced to the following problem:

$$\text{Maximize} \quad K(x) = (c_0 + \sum_{j=1}^{s} c_j x_j)/(d_0 + \sum_{j=1}^{s} d_j x_j)$$

$$\text{subject to} \sum_{j=1}^{s} a_j x_j \leq b ,$$

$$x_j; \text{ nonnegative integer, } j=1,2,\ldots,s.$$

Since

$$(\sum_{j=1}^{s} c_j x_j)/(\sum_{j=1}^{s} d_j x_j) = p$$

holds, where

$$p = c_1/d_1 = c_2/d_2 = \cdots = c_s/d_s (> c_0/d_0),$$

the objective function $K(x)$ is maximized when $\sum_{j=1}^{s} c_j x_j$ is maximized (by lemma 6.1). Thus the above problem is equivalent to (6.9).

(ii) Obvious from Theorem 6.3. □

It is known that a good feasible solution of the knapsack problem (6.9) (called the greedy solution) is available [Hu,T.C. and H.L. lenard,'73] [Magazine,M.J.et.al., '75] [Salkin,'75].

Greedy solution of the knapsack problem: Assume

$$c_1/a_1 \geq c_2/a_2 \geq \cdots \geq c_s/a_s \tag{6.10}$$

without loss of generality. The greedy solution $\bar{x} = (\bar{x}_1, \bar{x}_2, \cdots, \bar{x}_s)$ is given by

$$\bar{x}_1 = \left\lfloor b/a_1 \right\rfloor, \quad \bar{x}_{k+1} = \left\lfloor (b - \sum_{i=1}^{k} a_i \bar{x}_i)/a_{k+1} \right\rfloor, \quad k=1,\ldots,s-1. \tag{6.11}$$

Theorem 6.2 and 6.4, and above discussion suggest that the following "greedy solution" $\tilde{x} = (\tilde{x}_1, \tilde{x}_2, \cdots, \tilde{x}_n)$ of K is also a good feasible solution even if it is not optimal.

Greedy solution of K: Assume (6.2) and that $c_i/d_i = c_{i+1}/d_{i+1}$ implies $c_i/a_i \geq c_{i+1}/a_{i+1}$, $i=1,2,\ldots,n-1$ (see (6.10), and let

$$\tilde{x}_1 = b/c_1, \quad \tilde{x}_{k+1} = \left\lfloor (b - \sum_{i=1}^{k} a_i \tilde{x}_i)/a_{k+1} \right\rfloor, \quad k=1,\ldots,m-1$$

$$\tilde{x}_{m+1} = \tilde{x}_{m+2} = \cdots = \tilde{x}_n = 0, \tag{6.12}$$

where m is the smallest integer satisfying

$$(c_0 + \sum_{i=1}^{m} c_i \tilde{x}_i)/(d_0 + \sum_{i=1}^{m} d_i \tilde{x}_i) \geq c_{m+1}/d_{m+1} .$$

(Since an optimal solution satisfies $x_{m+1}=x_{m+2}=\cdots=x_n=0$ by Theorem 6.2, these variables may be deleted in the subsequent computation discussed in Section 6.6.)

6.5 Subsidiary Problem P(ξ) and Its Properties

Consider the following problem P(ξ) with parameter ξ.

$$\text{P}(\xi): \quad \text{Maximize} \quad Z(x) \triangleq N(x) - \xi D(x) = c_0 - \xi d_0 + \sum_{k=1}^{n} (c_k - \xi d_k) x_k$$

$$\text{subject to} \quad \sum_{k=1}^{n} a_k x_k \leqq b$$

$$x_k; \text{ nonnegative integer, } k=1,2,..,n.$$

Note that P(ξ) is the ordinary knapsack problem studied in the literature (e.g., [Gilmore and Gomory,'66] . [Garfinkel and Nemhauser,'72]). Let Z_ξ denote the optimal value of P(ξ) for each ξ. A close relation between K and P(ξ) has been pointed out by Jagannathan [Jagannathan,'66] and Dinkelbach [Dinkelbach,'67], and summarized in Theorem 5.1 and Theorem 5.2 in the previous Chapter 5. These are starting points of our algorithm for solving K discussed in Section 6.6.

Theorem 6.5. Let x^{ξ} and $x^{\xi'}$ be optimal solutions of P(ξ) and P(ξ') respectively, where $0 < \xi < \xi' \leqq \xi^*$ and $\xi^* = N(x^*)/D(x^*)$ for an optimal solution x^* of K. Then

$$K(x^{\xi}) \leqq K(x^{\xi'}) .$$

Proof: From the optimality of x^{ξ} and $x^{\xi'}$,

$$N(x^{\xi}) - \xi D(x^{\xi}) \geqq N(x^{\xi'}) - \xi D(x^{\xi'}) \qquad (6.13)$$

$$N(x^{\xi'}) - \xi' D(x^{\xi'}) \geqq N(x^{\xi}) - \xi' D(x^{\xi}) \qquad (6.14)$$

holds. Adding (6.13) to (6.14) and rearranging the result,

$$(\xi - \xi')[D(x^{\xi'}) - D(x^{\xi})] \geqq 0$$

follows. Thus $\xi < \xi'$ implies

$$D(x^{\xi'}) \leqq D(x^{\xi}) \ . \qquad (6.15)$$

Then $K(x^{\xi}) \leqq K(x^{\xi'})$ follows from (6.14) by dividing its left hand side by $D(x^{\xi'})$ and its right hand side by $D(x^{\xi})$, respectively. □

6.6 Algorithms for the Fractional Knapsack Problem

Dinkelbach [Dinkelbach,'67] proposed an algorithm which can be applied to rather general class of nonlinear fractional programming problems. His algorithm specified to the fractional knapsack problem K is described as follows.

Dinkelbach's algorithm

Step 1: (Initialize): Let $\xi_0 \leftarrow N(x^0)/D(x^0)$ for a feasible solution x^0 (e.g., $x^0 = (0, \cdots, 0)$) and let $\ell \leftarrow 0$. Go to Step 2.

Step 2: (Solve $P(\xi_\ell)$): Solve $P(\xi_\ell)$ and obtain the optimal value Z_{ξ_ℓ} and an optimal solution x^{ξ_ℓ}. If $Z_{\xi_\ell} = 0$, then x^{ξ_ℓ} is an optimal solution of K and ξ_ℓ is its value. Terminate. Otherwise, go to Step 3.

Step 3: (Increase): Let $\xi_{\ell+1} \leftarrow N(x^{\xi_\ell})/D(x^{\xi_\ell})$ and $\ell \leftarrow \ell+1$. Return to Step 2.

Note that various algorithms are available (e.g., [Gilmore and Gomory,'66] [Garfinkel and Nemhauser,'72]) to solve the ordinary knapsack problem $P(\xi_\ell)$ in Step 2.

$\xi_{\ell+1} > \xi_\ell$ always holds in Step 3 since $K(x^{\xi_\ell})$ $(=\xi_{\ell+1})$ $> \xi_\ell$ $(=K(x^{\xi_{\ell-1}})$ follows from Theorem 5.3 in Chapter 5 and the property that $Z_{\xi_\ell} = Z(x^{\xi_\ell}) > 0$ holds in $P(\xi_\ell)$. (Note that $Z(x^{\xi_\ell}) \neq 0$ since otherwise computation terminates in Step 2, and $Z(x^{\xi_\ell}) \geqq 0$ by $\xi_\ell = K(x^{\xi_{\ell-1}}) \leqq \xi^*$ and Theorem 5.1 (i)(ii).)

It is possible to solve K by this Algorithm. However it is modified in this chapter in an attempt to facilitate

the computation, based on the following Fact (1), (2).

Fact 1: A good feasible solution of K can be obtained with a small computational effort. It is given by (6.12) in Section 6.4.

Fact 2: A good feasible solution of knapsack problem $P(\xi)$ is also easily obtained by (6.11) in Section 6.4. An efficient computational method is also known to test a sufficient condition for the obtained solution to be an exact optimal solution of $P(\xi)$ ([Magazine,et.al.,'75]).

The feasible solution obtained in fact 1 above is used as an initial solution x^0 in the Dinkelbach's algorithm. To make use of Fact 2, it needs to modify it as follows. The basic idea is to use suboptimal solutions in place of optimal solutions of $P(\xi_\ell)$ as far as the improvement in ξ is attained. This is possible since any feasible solution x of $P(\xi_\ell)$ having $Z(x) > 0$ results in $\xi_{\ell+1}$ that is strictly greater than ξ_ℓ, by Theorem 5.3. (In fact, the purpose of obtaining x^{ξ_ℓ} in Step 2 is to find a feasible solution satisfying $Z(x) > 0$. Furthermore it is sometimes possible for nonoptimal solution of $P(\xi_\ell)$ to yield a larger $\xi_{\ell+1}$ than the one obtained from an optimal solution of $P(\xi_\ell)$.) If the greedy solution is not "accurate" enough to improve ξ, the exact optimal solution of $P(\xi_\ell)$ is obtained by one of the existing algorithms for the knapsack problem. It is expected that only a small number of requests to the routine for the exact optimal solution are necessary, in view of that the greedy solutions are usually very close to optimal. Even if the worst case, the number of requesting the routine does not

excced that of Dinkelbach's algorithm as shown in Theorem 6.7.

Modified Algorithm

Step 1: (Initialize): Let $\xi_0 \leftarrow N(\tilde{x})/D(\tilde{x})$ ($\tilde{x}$ is given by (6.12) in Section 6.4) and $\ell \leftarrow 0$. Go to Step 2.

Step 2: (Find the greedy solution of $P(\xi_\ell)$): Find $\bar{x}^{\xi_\ell}$, the greedy solution of $P(\xi_\ell)$ given by (6.11). Go to Step 5 if $Z(\bar{x}^{\xi_\ell}) < 0$; Step 4 if $Z(\bar{x}^{\xi_\ell}) = 0$; Step 3 if $Z(\bar{x}^{\xi_\ell}) > 0$.

Step 3: (Increase ξ): Let $\xi_{\ell+1} \leftarrow N(\bar{x}^{\xi_\ell})/D(\bar{x}^{\xi_\ell})$ and $\ell \leftarrow \ell+1$. Return to Step 2.

Step 4: (Test the optimality of $\bar{x}^{\xi_\ell}$): Test whether $\bar{x}^{\xi_\ell}$ is an optimal solution of $P(\xi_\ell)$ by Theorem 1 or Theorem 4 of [Magazine,et.al.,'75]. If optimal, $\bar{x}^{\xi_\ell}$ is also an optimal solution of K and its value is ξ_ℓ; terminate. Otherwise, go to Step 5.

Step 5: (Solve $P(\xi_\ell)$): Solve $P(\xi_\ell)$ and obtain an optimal solution x^{ξ_ℓ} and its value Z_{ξ_ℓ}. If $Z_{\xi_\ell} = 0$, then x^{ξ_ℓ} is an optimal solution of K and its value is ξ_ℓ; terminate. Otherwise (i.e., $Z_{\xi_\ell} > 0$), go to Step 6.

Step 6: (Increase ξ): Let $\xi_{\ell+1} \leftarrow N(x^{\xi_\ell})/D(x^{\xi_\ell})$ and $\ell \leftarrow \ell+1$. Return to Step 2.

Theorem 6.6. The Dinkelbach's algorithm and the modified algorithm terminate after solving a finite number of $P(\xi_\ell)$ (in Step 2 of Dinkelbach's algorithm, or in Step 5 of the modified algorithm), and provide an optimal solution of K upon termination.

Proof: Let x^{ξ_ℓ} be an optimal solution of $P(\xi_\ell)$ obtained

in Step 2 of the Dinkelbach's algorithm, or in Step 5 of the modified algorithm. Since

$$N(x^0)/D(x^0)(=\xi_0) < N(x^{\xi_0})/D(x^{\xi_0})(=\xi_1) < \cdots$$

holds, no solution repeatedly appears. This proves the finiteness of the algorithm since K has only a finite number of feasible solutions. When the termination is reached (i.e., $Z_{\xi_\ell} = 0$), then x^{ξ_ℓ} is an optimal solution of K by Theorem 5.1 (ii). □

Theorem 6.7. The total number of solving $P(\xi_\ell)$ in Step 5 of the modified algorithm does not exceed the total number of solving $P(\xi_\ell)$ in Step 2 of Dinkelbach's algorithm, provided that the same ξ_0 is used as the initial value.

Proof: Denote ξ of the ℓ-th $P(\xi)$ solved in Step 2 of the Dinkelbach's algorithm by ξ'_ℓ, and ξ of the ℓ-th $P(\xi)$ solved in Step 5 of the modified algorithm by ξ''_ℓ. (Note that $\xi''_\ell = \xi_\ell$(defined in the modified algorithm) does not usually hold since ℓ of ξ_ℓ counts also the number of greedy solutions obtained in Step 2). Now note $\xi'_0 = \xi''_0$ $(= \xi_0)$ by assumption. Since ξ''_0 may be improved by the greedy solution, it holds by Theorem 6.5 that $\xi'_1 \leq \xi''_1$. This in turn implies that $\xi'_2 \leq \xi''_2$. This argument can be repeated until $\xi'_p = \xi'_{p+1}$ $(\leq \xi''_p)$ holds (i.e., termination of the Dinkelbach's algorithm). Since $\xi^* = \xi'_p = \xi'_{p+1} \leq \xi''_p \leq \xi^*$ (ξ^* corresponds to an optimal solution of K), this shows that the modified algorithm terminates in at most (p+1) iterations. □

It is possible to derive an upper bound of the number of iterations required by the two algorithms. It will be given in Section 6.8, after discussing an example in Section 6.7.

Remark 6.3. Properties discussed in Section 6.2-6.5 may be used to facilitate the computation of the above algorithm.

(i) Some variables may be eliminated by Theorem 6.1, 6.3, 6.4, Corollary 6.2 and Remark 6.2, thereby reducing the problem size, prior to the execution of the algorithm.

(ii) A variable x_k can be fixed to 0 by Theorem 6.2, once $c_k - \xi d_k \leq 0$ (i.e., $K(x) = \xi \geq c_k/d_k$) holds for the current best solution x.

6.7 Example

By the modified algorithm, solve the following fractional knapsack problem K.

$$\text{K: Maximize } K(x) = \frac{2+5x_1+22x_2+17x_3+20x_4}{3+4x_1+18x_2+14x_3+18x_4}$$

$$\text{subject to } 5x_1+7x_2+4x_3+2x_4 \leq 19$$

$$x_1,\ x_2,\ x_3,\ x_4 \text{ ; nonnegative integers.}$$

This problem satisfies conditions (6.1), (6.2) and (6.3).

Step 1: The greedy solution $\tilde{x}$ of K given by (6.12) in Section 6.4 is

$$\tilde{x}_1 = \lfloor 19/5 \rfloor = 3,\ \tilde{x}_2 = \lfloor 4/6 \rfloor = 0,\ \tilde{x}_3 = 1,\ \tilde{x}_4 = 0.$$

$$\xi_0 \leftarrow K(\tilde{x}) = 34/29 \text{ and } \ell \leftarrow 0.$$

(Note that m=3 in (6.12) and x_4 may be omitted hereafter by Theorem 6.2.)

Step 2: $P(\xi_0)$: Maximize $(-44+9x_1+26x_2+17x_3)/29$

$$\text{subject to } 5x_1+7x_2+4x_3 \leq 19$$

$$x_1,\ x_2,\ x_3; \text{ nonnegative integers.}$$

The greedy solution $\bar{x}^{\xi_0}$ of $P(\xi_0)$ given by (6.11) is

$$\bar{x}_1^{\xi_0} = \lfloor 19/4 \rfloor = 4, \quad \bar{x}_2^{\xi_0} = \lfloor 3/7 \rfloor = 0, \quad \bar{x}_3^{\xi_0} = 0.$$

Its value $Z(\bar{x}^{\xi_0})$ is $24/29 > 0$, and Step 3 is entered.

Step 3: $\xi_1 \leftarrow K(\bar{x}^{\xi_0}) = 70/59$ and $\ell \leftarrow 1$. Return to Step 2.

Step 2:

$$P(\xi_1): \quad \text{Maximize} \quad (-92+15x_1+38x_2+23x_3)/59$$

$$\text{subject to} \quad 5x_1+7x_2+4x_3 \leqq 19$$

$$x_1, x_2, x_3; \text{ nonnegative integers.}$$

The greedy solution is again $\bar{x}^{\xi_1}=(0,0,4)$ $(=\bar{x}^{\xi_0})$ and its value $Z(\bar{x}^{\xi_1})$ is 0. Go to Step 4.

Step 4: Test given in [Magazine, et.al.,'75] fails to prove the optimality of $\bar{x}^{\xi_1}$. Thus Step 5 is entered.

Step 5: An optimal solution x^{ξ_1} of $P(\xi_1)$ is obtained;

$$x^{\xi_1} = (0, 1, 3) \text{ and } Z(x^{\xi_1}) = 15/59 > 0.$$

Go to Step 6.

Step 6: $\xi_2 \leftarrow K(x^{\xi_1})=25/21$ and $\ell \leftarrow 2$. Return to Step 2.

Step 2:

$$P(\xi_2): \quad \text{Maximize} \quad (-33+5x_1+12x_2+7x_3)/21$$

$$\text{subject to} \quad 5x_1+7x_2+4x_3 \leqq 19$$

$$x_1, x_2, x_3; \text{ nonnegative integers.}$$

The greedy solution $\bar{x}^{\xi_2}$ is again (0,0,4) and $Z(\bar{x}^{\xi_2})$ < 0 by Theorem 6.7 applied to $P(\xi_2)$. Go to Step 5.

Step 5: An optimal solution $\bar{x}^{\xi_2}$ of $P(\xi_2)$ is obtained;

$$x^{\xi_2} = (0, 1, 3) \text{ and } Z(x^{\xi_2}) = 0.$$

This is an optimal solution of K and its value is $\xi_2 = 25/21$.

6.8 Upper Bound on the Number of Iterations

This section gives an upper bound on the number of $P(\xi_\ell)$ whose exact optimal solutions are necessary (in Step 2 of the Dinkelbach's algorithm, or in Step 5 of the modified algorithm). It is given by $min[A,B]$ where

$$A = \max_{1 \leq \ell \leq n} \lfloor c_\ell b/a_\ell \rfloor - (g_1 c_1 + g_2 c_2) + 2 \tag{6.16}$$

$$B = \max_{1 \leq \ell \leq n} \lfloor d_\ell b/a_\ell \rfloor - (g_1 d_1 + g_2 d_2) + 2 \tag{6.17}$$

$$g_1 = \lfloor b/a_1 \rfloor \text{ and } g_2 = \left\lfloor (b - \lfloor b/a_1 \rfloor a_1)/a_2 \right\rfloor .$$

This is now proved via four lemmas. In the following, it is assumed that

$$(c_0 + c_1 g_1)/(d_0 + d_1 g_1) < c_2/d_2 , \tag{6.18}$$

since otherwise an optimal solution satisfies $x_2 = x_3 = \cdots = x_n = 0$ by Theorem 6.2, and $\hat{x}$ given by (6.7) is optimal.

Lemma 6.4. Under assumption (6.18), an optimal solution x^* of K satisfies

$$\sum_{k=1}^{n} c_k x_k^* \geq g_1 c_1 + g_2 c_2 . \tag{6.19}$$

Proof: First note that $\bar{x} = (g_1, g_2, 0, \cdots, 0)$ is feasible. The proof is done by considering two cases.

Case (i): $(c_1g_1+c_2g_2)/(d_1g_1+d_2g_2) \geq (\sum_{k=1}^{n} c_k x_k^*)/(\sum_{k=1}^{n} d_k x_k^*)$

$(>c_0/d_0)$ (6.20)

By assumption (6.20) and the optimality of x^*, Lemma 6.2 implies that

$$\sum_{k=1}^{n} c_k x_k^* \geq c_1g_1 + c_2g_2 \quad .$$

Case (ii): $(\sum_{k=1}^{n} c_k x_k^*)/(\sum_{k=1}^{n} d_k x_k^*) > (c_1g_1+c_2g_2)/(d_1g_1+d_2g_2).$ (6.21)

Assumption (6.21) implies by (6.2) and Lemma 6.1 that $x_1^*/x_2^* > g_1/g_2$, and hence

$$x_2^* < g_2 \tag{6.22}$$

by the greediness of g_1 (i.e., $g_1 \geq x_1^*$). Then

$$K(\bar{x}) = \frac{c_0+c_1x_1^*+c_2x_2^*+(g_1-x_1^*)c_1+(g_2-x_2^*)c_2}{d_0+d_1x_1^*+d_2x_2^*+(g_1-x_1^*)d_1+(g_2-x_2^*)d_2}$$

$$\leq K(x^*) = \frac{c_0 + c_1x_1^* + c_2x_2^* + \sum_{k=3}^{n} c_k x_k^*}{d_0 + d_1x_1^* + d_2x_2^* + \sum_{k=3}^{n} d_k x_k^*} \tag{6.23}$$

implies

$$(\sum_{k=3}^{n} c_k x_k^*)/(\sum_{k=3}^{n} b_k x_k^*) > (c_0+c_1x_1^*+c_2x_2^*)/(d_0+d_1x_1^*+d_2x_2^*)$$

since, otherwise

$$K(x^*) \leq (c_0+c_1x_1^*+c_2x_2^*)/(d_0+d_1x_1^*+d_2x_2^*)$$

(by $(\sum_{k=3}^{n} c_k x_k^*)/(\sum_{k=3}^{n} d_k x_k^*) \leq K(x^*)$ and Lemma 6.3)

$$\leq (c_0+c_1g_1+c_2x_2^*)/(d_0+d_1g_1+d_2x_2^*)$$

(by Lemma 6.1, (6.22) and greediness of g_1)

$$< c_2/d_2 \quad \text{(by Lemma 6.1 and (6.18))} \qquad (6.24)$$

and (6.24) implies

$$K(\bar{x}) = (c_0+c_1g_1+c_2g_2)/(d_0+d_1g_1+d_2g_2)$$

$$= (c_0+c_1g_1+c_2x_2^*+c_2(g_2-x_2^*))/(d_0+d_1g_1+d_2x_2^*+d_2(g_2-x_2^*))$$

$$> (c_0+c_1g_1+c_2x_2^*)/(d_0+d_1g_1+d_2x_2^*)$$

(by (6.18) and Lemma 6.1)

$$\geq K(x^*),$$

which is a contradiction to (6.23). Considering

$$(c_0+c_1x_1^*+c_2x_2^*)/(d_0+d_1x_1^*+d_2x_2^*)<(\sum_{k=3}^{n} c_kx_k^*)/(\sum_{k=3}^{n} d_kx_k^*)$$

$$\leq ((g_1-x_1^*)c_1+(g_2-x_2^*)c_2)/((g_1-x_1^*)d_1+(g_2-x_2^*)d_2) \quad (6.25)$$

(by (6.2) and Lemma 6.1),

as $p/q<r/s\leq t/u$ of Lemma 6.2, we have immediately

$$t\leq r \text{ , i.e., } (g_1-x_1^*)c_1+(g_2-x_2^*)c_2 \leq \sum_{k=3}^{n} c_kx_k^* \text{ .} \quad (6.26)$$

Thus

$$c_1g_1 + c_2g_2 \leq \sum_{k=1}^{n} c_kx_k^*$$

is proved. □

Lemma 6.5. If (6.18) holds, then an optimal solution x^* of K satisfies

$$\sum_{k=1}^{n} d_kx_k^* \geq d_1g_1 + d_2g_2 \text{ .}$$

Proof: Assume

$$\sum_{k=1}^{n} d_kx_k^* < d_1g_1 + d_2g_2 \text{ .} \quad (6.27)$$

Then $\sum_{k=1}^{n} c_kx_k^* \geq c_1g_1 + c_2g_2$ implies

$$(\sum_{k=1}^{n} c_kx_k^*)/(\sum_{k=1}^{n} d_kx_k^*) > (c_1g_1+c_2g_2)/(d_1g_1+d_2g_2), \quad (6.28)$$

which is equal to (6.21). Thus Case(ii) in the proof of Lemma 6.4 is applied. However, (6.27) (i.e., $(g_1-x_1^*)d_1 + (g_2-x_2^*) > \sum_{k=3}^{n} d_k x_k^*$) and (6.26) lead to a contradiction to (6.25). □

Lemma 6.6. Let ξ^* be the optimal value of K and x^{ξ} be an optimal solution of $P(\xi)$ for ξ, where $\xi^* > \xi > 0$. Moreover, let $x^{\xi'}$ be an optimal solution of $P(\xi')$ for $\xi' = K(x^{\xi})$ (i.e., $\xi' > \xi$ by Theorem 5.1 (i) and Theorem 5.3 in Chapter 5). Then the following properties hold.

(i) If $\xi^* > \xi'$, then $D(x^{\xi'}) < D(x^{\xi})$ and $N(x^{\xi'}) < N(x^{\xi})$.

(ii) If $\xi^* = \xi'$. then $D(x^{\xi'}) = D(x^{\xi})$ and $N(x^{\xi'}) = N(x^{\xi})$, or

$D(x^{\xi'}) < D(x^{\xi})$ and $N(x^{\xi'}) < N(x^{\xi})$.

Proof: First note that $D(x^{\xi'}) \leq D(x^{\xi})$ holds by (6.15).

Case (a): $D(x^{\xi'}) = D(x^{\xi})$. Then $N(x^{\xi}) = N(x^{\xi'})$ follows from (6.13) and (6.14), i.e., $K(x^{\xi}) = K(x^{\xi'})$.

This implies $Z_{\xi'} = 0$ and hence $\xi^* = \xi'$ holds by Theorem 5.1 (ii) in Chapter 5.

Case (b): $D(x^{\xi'}) < D(x^{\xi})$. From (6.13), this implies $N(x^{\xi'}) < N(x^{\xi})$. Case (a) and Case (b) together prove (i) (ii) of Lemma 6.6. □

Lemma 6.7. Let x^* be an optimal solution of K. Then

$$\sum_{k=1}^{n} c_k x_k^* \leq \max_{1 \leq \ell \leq n} \left\lfloor c_\ell b/a_\ell \right\rfloor,$$

$$\sum_{k=1}^{n} d_k x_k^* \leq \max_{1 \leq \ell \leq n} \left\lfloor d_\ell b/a_\ell \right\rfloor.$$

Proof: Consider the following linear programming problem:

$$\text{Maximize } h = \sum_{k=1}^{n} c_k x_k$$

$$\text{subject to } \sum_{k=1}^{n} a_k x_k \leq b$$

$$x_k \geq 0, \quad k=1,2,\ldots,n.$$

It is known that the optimal value h^* of this problem is given by

$$h^* = \max_{1 \leq \ell \leq n} c_\ell b/a_\ell .$$

Since c_k, d_k are all assumed to be integers and only integer solutions are sought in K, any feasible solution x of K satisfies

$$\sum_{k=1}^{n} c_k x_k \leq \lfloor h^* \rfloor = \max_{1 \leq \ell \leq n} \lfloor c_\ell b/a_\ell \rfloor .$$

$\sum_{k=1}^{n} d_k x_k^*$ may be similarly treated. □

From Lemma 6.4-6.7, the following theorem is proved.

Theorem 6.8. Under assumption (6.18), the number of executions of Step 2 of the Dinkelbach's algorithm or Step 5 of the modified algorithm does not exceed

$$\min[\, A, B] \ ,$$

where A and B are given in (6.16) and (6.17).

Proof: Consider the Dinkelbach's algorithm. By Lemma 6.7,

$$\sum_{k=1}^{n} c_k x_k \leqq \max_{1 \leqq \ell \leqq n} \left\lfloor c_\ell b / a_\ell \right\rfloor$$

holds for a feasible solution x of K. Lemma 6.6 (i) shows, if none of optimal solutions of $P(\xi_\ell)$ and $P(\xi_{\ell+1})$ is an optimal solution of K, that

$$\sum_{k=1}^{n} c_k x_k^{\xi_\ell} > \sum_{k=1}^{n} c_k x_k^{\xi_\ell}$$

holds. Thus $\sum_{k=1}^{n} c_k x_k$ decreases at least by unity since it is an integer. Repeating this argument, a decreasing sequence

$$\sum_{k=1}^{n} c_k x_k^{\xi_0} > \sum_{k=1}^{n} c_k x_k^{\xi_1} > \cdots > \sum_{k=1}^{n} c_k x_k^{\xi_p} \geqq \sum_{k=1}^{n} c_k x_k^{*}$$

is derived. Since

$$\sum_{k=1}^{n} c_k x_k \geqq c_1 g_1 + c_2 g_2$$

by Lemma 6.4, it is proved that at most A executions of Step 2 are required. By applying a similar argument to $\sum_{k=1}^{n} d_k x_k$, it is shown that at most

$$\min[\, A, B]$$

executions of Step 2 in the Dinkelbach's algorithm are necessary.

As for the modified algorithm, the number of executions of Step 5 is not more than that of Step 2 of the Dinkelbach's algorithm by Theorem 6.7. □

6.9 Conclusion

This chapter investigated the fractional knapsack problems and proposed a modified algorithm of Dinkelbach's. The algorithm exploits the fact that good feasible solutions are easily obtained for both the fractional knapsack problem and the ordinary knapsack problem $P(\xi)$ generated as a subsidiary problem with a parameter ξ. The technique in this chapter that uses a good feasible solution of each subsidiary problem to update ξ, seems to be extensible to other fractional programming problems, e.g., fractional version of set covering problems. Moreover, this chapter has clarified some new theoretical properties of fractional programming problems. Among them, an upper bound of the number of iterations and the monotone decrease of $N(x^{\xi})$ and $D(x^{\xi})$ as parameter ξ is increased seems to be particularly useful for constructing efficient algorithms for fractional programming problems. In addition this chapter has clarified the behavior of optimal solutions when the right hand side b of the constraint is increased.

Generally speaking, properties of fractional programming problems have not been well explored yet, except for the case of linear fractional programming problems. Therefore, clarifying general properties of fractional programming problems is a matter of great urgency.

CHAPTER 7 QUADRATIC FRACTIONAL PROGRAMMING PROBLEMS

7.1 Introduction

This chapter discusses quadratic fractional programming problem in which both numerator and denominator of the fractional objective function are quadratic functions of continuous variables, and its feasible region is a polyhedron defined by linear inequalities. This type of problems may be encountered for example in the maximum probability model of stochastic linear programming, where the numerator of the objective function is specified by the expectation of some random variables and the denominator by variance of some random variables ([Charnes and Cooper,'63] [Geoffrion,'67b]).

The properties of the quadratic fractional programming problem have much in common with the discrete fractional programming problems discussed in Chapter 5 and Chapter 6. The main purpose of this chapter is to extend the solution procedures developed for discrete fractional programming problems to the quadratic one. Two algorithms are proposed. The first one is a straightforward application of the parametric programming approach. The second is a modification of Dinkelbach method.

Section 7.2 gives the definition of the quadratic fractional programming problem and introduces the subsidiary problem which is the ordinary quadratic programming problem. Section 7.3 proposes algorithm A based on the idea of parametric programming. Section 7.4 also includes n analysis of basis matrix in the linear complementary

equation defined from the Kuhn-Tucker condition for the subsidiary problem. The observed properties are used to prove the finiteness of the algorithm. Section 7.5 discusses two special cases of the problem, and it is shown that the results in Section 7.4 can be considerably simplified. Section 7.6 deals with Algorithm B, a modification of Dinkelbach method, and proves its finiteness. Section 7.7 compares Algorithm A and Algorithm B, based on some computational experience on computers. Final Section 7.8 concludes this chapter.

7.2 Formulation of Quadratic Fractional Programming Problems

This chapter discusses the following problem QF, as a special case of (5.1) in which N(x) and D(x) are quadratic functions, and S is a polyhedron defined by a set of linear inequalities.

Problem QF: Maximize $(\frac{1}{2}x'Cx + r'x + s)/(\frac{1}{2}x'Dx + p'x + q)$

$$\text{subject to} \quad Ax \leqq b \ , \tag{7.1}$$

where C is n×n negative definite matrix; D is an n×n positive semidefinite matrix; A is an m×n matrix; r, p are n-vectors; b is a m-vector and x is a n-vector of variables. All coefficients in the matrices and the vectors are reals. It is assumed that

$$\begin{aligned} &\tfrac{1}{2}x'Cx + r'x + s \geqq 0 \quad \text{for some } x \in S \\ &\tfrac{1}{2}x'Dx + p'x + q > 0 \quad \text{for some } x \in S \end{aligned} \tag{7.2}$$

where $S = \{x \mid Ax \leqq b\}$. Corresponding to Problem QF, the subsidiary problem Q(ξ) (also special case of P(ξ) of Section 5.2) is written as follows:

Q(ξ): Maximize $Z(\xi) \triangleq \frac{1}{2}x'H(\xi)x + c(\xi)'x + d(\xi)$

$$\text{subject to} \quad Ax \leqq b \tag{7.3}$$

where $H(\xi) = C - \xi D$, $c(\xi) = r - \xi p$ and $d(\xi) = s - \xi q$.

Note that finite algorithms for solving quadratic programming problem $Q(\xi)$ are known ([Wolfe,P.,'59] [Hadley, G.,'64]) since $H(\xi)$ is negative definite for $\xi \geq 0$. In other words, Z_ξ for a given ξ can be calculated in finite steps. Then QF is solved by finding $\xi = \xi^*$ such that $Z_{\xi^*}=0$ from the results of Section 5.2. The search for ξ^* would be facilitated by resorting to the parametric programming technique developed for the quadratic programming.

7.3 Algorithm Based on Parametric Programming

The Kuhn-Tucker Theorem (e.g., [Hadley,'64]) shows that an optimal solution of $Q(\xi)$ is obtained by solving the following problem $Q'(\xi)$.

$$Q'(\xi): \quad \overbrace{\begin{bmatrix} H(\xi) & A' & \mathbf{0} \\ A & \mathbf{0} & I_m \end{bmatrix}}^{n+2m} \Big\}\, n+m \quad \begin{pmatrix} x \\ u \\ y \end{pmatrix} = \begin{pmatrix} -c(\xi) \\ b \end{pmatrix} \tag{7.4}$$

$$y'u = 0, \quad y \geqq 0, \quad u \leqq 0, \tag{7.5}$$

where I_m is the unit matrix of order m, and y, u are m-vectors of variables.

Now let B be a basis of (7.4) (i.e., (n+m)×(n+m) nonsingular submatrix of the matrix in (7.4) and let x_B, u_B, y_B be the basic variables corresponding to columns of B. Then

$$\begin{pmatrix} x_B \\ u_B \\ y_B \end{pmatrix} = B^{-1} \begin{pmatrix} -c(\xi) \\ b \end{pmatrix} \tag{7.6}$$

nonbasic variables = 0,

is the basic solution corresponding to B. The solution (or B) is said feasible if (7.5) is also satisfied. It is known that x_B of any feasible solution (7.6) is an optimal solution of $Q(\xi)$. For a feasible B for $\xi = \xi_i$, the interval $[\xi_i, \xi_{i+1}]$ exists such that ξ_{i+1} is the

maximum to keep B feasible for all $\xi \in [\xi_i, \xi_{i+1}]$. Thus the curve Z_ξ as illustrated in Figure 7.1 is partitioned into subintervals, each of which corresponds to one basis B. (In Figure 7.1, a small circle indicates a point of basis change.)

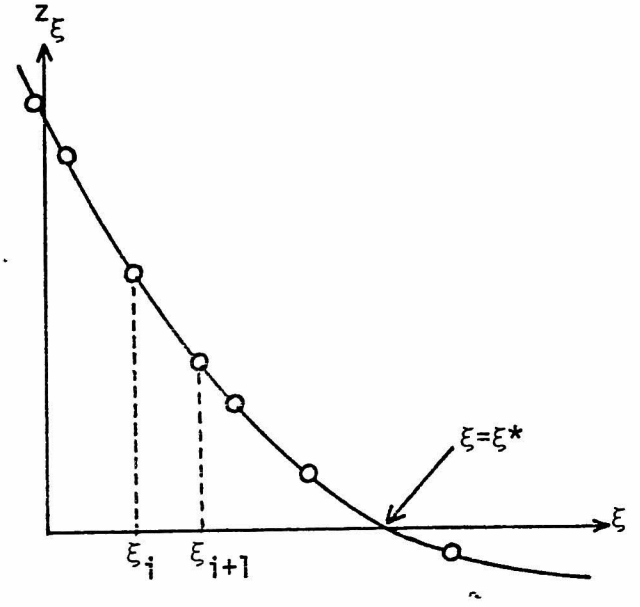


Figure 7.1. z_ξ vs. ξ.

Now we have the following algorithm for QF.

Algorithm A

Step A4: $i \leftarrow 0$ and $\xi_i \leftarrow 0$ (or an appropriate value such that $Z_{\xi_0} \geq 0$).

Step A2: Solve $Q'(\xi_i)$ (this implies to solve $Q(\xi_i)$). If

$Q'(\xi_i)$ is infeasible, so is QF; halt. Otherwise obtain the maximum ξ_{i+1} such that $\xi_{i+1} \geqq \xi_i$ and $Q'(\xi)$ has the same feasible basis for all $\xi \in [\xi_i, \xi_{i+1}]$.

Step A3: If $Z_{\xi_{i+1}} \leqq 0$ or $\xi_{i+1} = \infty$, go to Step A4; otherwise let $i \leftarrow i+1$ and return to Step A2.

Step A4: Calculate $\xi^* \in [\xi_i, \xi_{i+1}]$ such that $Z_\xi|_{\xi=\xi^*} = 0$, and halt. ξ^* is the optimal value of QF.

Remark 7.1. $Q'(\xi_i)$ is solved in finite steps for example by the Wolfe's method [Wolfe,'59] [Hadley,'64]: (The Wolfe's method is used in the experiment of Section 7.7.) Given an optimal tableau for $Q'(\xi_i)$, an optimal tableau for $Q'(\xi_{i+1})$ is easily obtained by using the parametric programming technique developed by Ritter [Ritter,'67] and Wolfe [Wolfe,'59] (Wolfe treats the case of $D = 0$) in one pivot operation (provided that the nondegeneracy assumption holds). It is not necessary to solve $Q'(\xi_{i+1})$ from the initial tableau.

Remark 7.2. ξ_{i+1} in Step A2 is obtained from (7.6) by calculating the maximum of ξ' ($\geqq \xi_i$) such that the basic feasible solution (7.6) is feasible for all ξ satisfying $\xi_i \leqq \xi \leqq \xi'$. (See Example 7.1 of Section 7.4 in which this process is carried out.) If $D = 0$, this computation becomes particularly simple (as studied by Wolfe [Wolfe,'59]) since B^{-1} is independent of ξ. ($D = 0$ is assumed in the computational experiment in Section 7.7.)

Remark 7.3. ξ^* in Step A4 is the solution of equation

$$\frac{1}{2} x'H(\xi)x + c(\xi)'x + d(\xi) = 0 \qquad (7.7)$$

which is the smallest among ξ not smaller than ξ_i, where x is given by (7.6). If $D = 0$, (7.7) is a quadratic equation, and the smaller of two solutions is ξ^*.

Remark 7.4. An optimal solution of QF is easily obtained from (7.6) by setting $\xi = \xi^*$.

7.4 Finiteness of Algorithm A

Provided that a finite algorithm is used to solve $Q'(\xi_i)$ in Step A2 (see Remark 7.1), algorithm A is proved to be finite if the number of intervals of Z_ξ each of which corresponds to a basis (see Section 7.3) is finite. Since the number of possible bases of (7.4) is finite, it then suffices to show that the same basis B appears only finitely many times corresponding to different intervals.

Note that the finiteness is not trivial since there is a case in which the same basis corresponds to more than one interval, as given in the next example. It also helps to visualize the idea used in the proof for the finiteness in the latter half of this section.

Example 7.1.

$$\text{QF:} \quad \text{Maximize} \quad \frac{\frac{1}{2}(x_1, x_2)\begin{bmatrix}-1 & -1\\ 1 & -2\end{bmatrix}\begin{pmatrix}x_1\\ x_2\end{pmatrix} + (1, -1)\begin{pmatrix}x_1\\ x_2\end{pmatrix} + 180}{\frac{1}{2}(x_1, x_2)\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\begin{pmatrix}x_1\\ x_2\end{pmatrix} + (-1,-7)\begin{pmatrix}x_1\\ x_2\end{pmatrix} + 1}$$

$$\text{subject to} \quad \begin{bmatrix}1 & 1\\ -1 & 0\\ 0 & -1\end{bmatrix}\begin{pmatrix}x_1\\ x_2\end{pmatrix} \leq \begin{pmatrix}7\\ 0\\ 0\end{pmatrix} .$$

The objective function of $Q(\xi)$ is then given by

$$Z(x)=(x_1,x_2)\begin{bmatrix}-1-\xi & -1\\ 1 & -2-\xi\end{bmatrix}\begin{pmatrix}x_1\\ x_2\end{pmatrix}+(1+\xi,-1+7\xi)\begin{pmatrix}x_1\\ x_2\end{pmatrix}+180-\xi.$$

Constraint (7.40) of the Kuhn-Tucker Theorem is

$$\begin{bmatrix}-1-\xi & -1 & 1 & -1 & 0 & \\ 1 & -2-\xi & 1 & 0 & -1 & \boldsymbol{0}\\ 1 & 1 & & & & \\ -1 & 0 & & \boldsymbol{0} & & I_3\\ 0 & -1 & & & & \end{bmatrix}\begin{pmatrix}x_1\\ x_2\\ u_1\\ u_2\\ u_3\\ y_1\\ y_2\\ y_3\end{pmatrix}=\begin{pmatrix}-1-\xi\\ 1-7\xi\\ 7\\ 0\\ 0\end{pmatrix}.$$

Now take the following basis:

$$B=\begin{bmatrix}-1-\xi & -1 & \\ 1 & -2-\xi & \boldsymbol{0}\\ 1 & 1 & \\ -1 & 0 & \\ 0 & -1 & I_3\end{bmatrix}. \qquad (7.8)$$

B^{-1} is then given by

$$B^{-1}=\begin{bmatrix}(-2-\xi)/\Delta & 1/\Delta & \\ -1/\Delta & (-1-\xi)/\Delta & \boldsymbol{0}\\ (3+\xi)/\Delta & \xi/\Delta & \\ (-2-\xi)/\Delta & 1/\Delta & \\ -1/\Delta & (-1-\xi)/\Delta & I_3\end{bmatrix}$$

where

$$\Delta=\det B=3+3\xi+\xi^2.$$

Note that $x_B = (x_1, x_2)'$, $u_B = \phi$ and $y_B = (y_1, y_2, y_3)'$, and

$$\begin{pmatrix} x_B \\ y_B \end{pmatrix} = \begin{Bmatrix} (3-4\xi+\xi^2)/\Delta \\ (7\xi+7\xi^2)/\Delta \\ (18+18\xi-\xi^2)/\Delta \\ (3-4\xi+\xi^2)/\Delta \\ (7\xi+7\xi^2)/\Delta \end{Bmatrix}$$

Since $\Delta > 0$ for any ξ, and

$$\begin{aligned} y_1 &\geqq 0 \text{ for } -0.95 \leqq \xi \leqq 18.95 \\ y_2 &\geqq 0 \text{ for } \xi \leqq 1 \text{ or } \xi \geqq 3 \\ y_3 &\geqq 0 \text{ for } \xi \leqq 1 \text{ or } \xi \geqq 0 , \end{aligned}$$

(7.5) is satisfied in two intervals:

$$[-0.95, 1] \text{ and } [3, 18.75] .$$

When Algorithm A starts from $\xi_0 = 0$, threfore, basis B of (7.8) appears twice corresponding to intervals

$$[0, 1] \text{ and } [3, 18.95].$$

(Note that $Z_{\xi_i} > 0$ still holds for $\xi_i = 3$.)

It was first proved by Ritter [Ritter,'67] in a more general setting that a basis B appears only finitely many times as feasible basis when ξ is continuously increased.

In the following, the same result is proved by de-

riving an explicit upper bound (not given in [Ritter,'67]) on how many times a basis B appears in Step A2, by refining the argument used in Section 3 and Section 4 of [Ritter,'67].

It is known in the theory of quadratic programming (e.g., [Ritter,'67]) that condition $y'u = 0$ of (7.5) permits us to consider only a basic solution with basis of the form

$$B = \begin{bmatrix} \overbrace{H(\xi)}^{n} & \overbrace{A_1'}^{k} & \overbrace{\mathbf{0}}^{m-k} \\ A_1 & \mathbf{0} & \mathbf{0} \\ A_2 & \mathbf{0} & I_{m-k} \end{bmatrix} \begin{matrix} \} n \\ \} k \\ \} m-k \end{matrix} \tag{7.9}$$

where $A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$, and $(k \times n)$ matrix A_1 has full rank. Thus we consider in the subsequent discussion only bases in this form. Since $H(\xi)$ is assumed to be negative definite,

$$(-1)^n \det H(\xi) > 0 \tag{7.10}$$

holds for any ξ.

Lemma 7.1. B of (7.9) satisfies

$$(-1)^n \det B > 0$$

for any ξ. (Thus det B does not change its sign when ξ is continuously increased.)

Proof:

$$(-1)^n \det B = (-1)^n \det \begin{bmatrix} H(\xi) & A_1' \\ A_1 & \boldsymbol{O} \end{bmatrix} \det I_{m-k} \quad \text{(by (7.9))}$$

$$= (-1)^n \det \begin{bmatrix} I_n & \boldsymbol{O} \\ A_1 H(\xi)^{-1} & I_k \end{bmatrix} \begin{bmatrix} H(\xi) & \boldsymbol{O} \\ \boldsymbol{O} & -A_1 H(\xi)^{-1} A_1' \end{bmatrix} \begin{bmatrix} I_n & H(\xi)^{-1} A_1' \\ \boldsymbol{O} & I_k \end{bmatrix}$$

$$= (-1)^n \det H(\xi) \det[-A_1 H(\xi)^{-1} A_1] > 0.$$

The last relation follows from (7.10) and the property that $-A_1 H(\xi)^{-1} A_1'$ is positive definite (hence $\det[-A_1 H(\xi)^{-1} A_1'] > 0$.) □

Lemma 7.2. Let u_i and y_j be elements of u_B and y_B defined in (7.6) for B of (7.9). Then the number of intervals in which u_i assumes nonpositive values when ξ is changed from 0 to ∞ is at most $\lceil (n-k+2)/2 \rceil$, where $\lceil w \rceil$ denotes the smallest integer not smaller than w. Similarly, the number of intervals in which y_j assumes nonnegative values is at most $\lceil (n-k+1)/2 \rceil$.

Proof: From (7.6), we have

$$\begin{pmatrix} x_B \\ u_B \\ y_B \end{pmatrix} = B^{-1} \begin{pmatrix} -c(\xi) \\ b \end{pmatrix} \tag{7.11}$$

$$= \begin{pmatrix} \Delta_{11}/\Delta & \Delta_{21}/\Delta & \cdots & \Delta_{n+m1}/\Delta \\ \vdots & & & \vdots \\ \Delta_{1n+m}/\Delta & \cdots\cdots\cdots & & \Delta_{n+m\,n+m}/\Delta \end{pmatrix} \begin{pmatrix} -c(\xi) \\ b \end{pmatrix} \qquad (7.11)$$

where Δ_{ij} is the (i, j)-th cofactor of B and $\Delta = \det B$. Since each element of $H(\xi)$ is linear in ξ and other elements of B do not depend on ξ, the degree (in ξ) of the numerator of each element of B^{-1} is easily calculated; it is given below.

$$\begin{array}{c|c|c} \overbrace{}^{n} & \overbrace{}^{k} & \overbrace{}^{m-k} \\ \left.\begin{array}{c|c|c} (n-k-1) & (n-k) & (0) \\ \hline (n-k) & (n-k+1) & (0) \\ \hline (n-k-1) & (n-k) & (0) \end{array}\right. & \begin{array}{c} \}\,n \\ \}\,k \\ \}\,m-k \end{array} \end{array} \qquad (7.12)$$

Note that only the numerator is important from the view point of the sign, since the denominator Δ does not change its sign by Lemma 7.1. From (7.12), the degrees of the numerators of x_B , u_B, y_B are obtained, using that $c(\xi)$ is linear in ξ.

$$\begin{pmatrix} x_B \\ u_B \\ y_B \end{pmatrix} : \begin{pmatrix} (n-k) \\ (n-k+1) \\ (n-k) \end{pmatrix} \begin{matrix} \}\,n \\ \}\,k \\ \}\,m-k \end{matrix} \qquad (7.13)$$

Now note that the number of intervals in which a polynomial of degree r assumes nonnegative (nonpositive) values is at most $\lceil (r+1)/2 \rceil$. By (7.13), this proves the lemma statement. □

Theorem 7.1. Let B be given by (7.9). When ξ is increased from 0 to ∞, the number of intervals of ξ in which u_B and y_B of basic solution (7.6) satisfy $u_B \leqq 0$ and $y_B \geqq 0$ (i.e., feasible) is at most

$$N(k) = k\left\lceil\frac{n-k+2}{2}\right\rceil + (m-k)\left\lceil\frac{n-k+1}{2}\right\rceil - (m-1). \qquad (7.14)$$

Proof: Let $p_i(\xi)$ be a polynomial of degree r_i, i=1,2, ...,m. Then it is easy to show that the number of intervals of ξ in which $p_1(\xi),\cdots,p_k(\xi)$ are nonpositive and $p_{k+1}(\xi),\cdots, p_m(\xi)$ are nonnegative is at most

$$\sum_{i=1}^{m}\left\lceil\frac{r_i+1}{2}\right\rceil - (m-1) \quad . \qquad (7.15)$$

(7.14) follows from (7.15) by substituting

$$r_i = \begin{cases} n-k+1 \ , & i=1,2,\ldots,k \\ n-k \ , & i=k+1,\ldots,m \end{cases}$$

obtained in Lemma 7.2. □

Corollary 7.2. $N(n) = 1$.

A basis B with k=n represents an extreme point of polyhedron $Ax \leqq b$. This corollary tells that such basis appears at most once as a feasible basis in Algorithm A.

Corollary 7.3. $N(0) = m\left\lceil\frac{n+1}{2}\right\rceil + 1-m$.

Note that basis B with k=0 corresponds to an interior point of polyhedron $Ax \leqq b$.

Theorem 7.2. Algorithm A of Section 7.3 gives an optimal solution of QF or indicates its infeasibility, after executing Step A2 finite times, provided that $Q'(\xi_i)$ is solved by finite algorithm.

Proof: The finiteness follows from the argument given in the beginning of this section and Theorem 7.1. If Algorithm A halts in Step A2 indicating the infeasibility of $Q'(\xi_i)$, then $Ax \leqq b$ is vacuous (i.e., QF is infeasible) . since it is known that a quadratic programming problem with a negative definite objective function (such as $Q(\xi_i)$) always has an optimal solution if $Ax \leqq b$ is not vacuous. On the other hand, if Algorithm A halts in Step A4, ξ^* is the maximum value of QF as proved in [Dinkelbach,'67] [Jagannathan,'66]. □

7.5 Two Special Cases

In this section, two special cases $D = 0$ and $C = 0$ are discussed. In either case, it is shown that each basis B appears at most once in Algorithm A.

Theorem 7.3. Assume $D = 0$ in QF. Then a basis B of (7.9) appears at most once as a feasible basis of $Q'(\xi)$ in Step A2 of Algorithm A.

Proof: In this case, no element of B of (7.9) contains ξ since $H(\xi) = C - \xi D = C$; hence no element of B^{-1} contains ξ. Thus x_B, u_B, y_B given by (7.6) is linear in variable ξ, since $-c(\xi)$ is linear in ξ. Letting $r_i = 1$ in (7.15) gives $N(k) = 1$ in the proof of Theorem 7.1. □

As mentioned in Remarks given to Algorithm A, the case of $D = 0$ has also other computational advantages. The computational experiment in Section 7.7 is therefore done for this simple case only.

The case of $C = 0$ is similar. However, it is necessary to assume that

$$D \text{ is positive definite} \tag{7.16}$$

$$r'x + s > 0 \text{ for some } x \in S \tag{7.17}$$

in addition to (7.2), since C is not negative definite in this case. Algorithm A should be started from $\xi_0 = \eta$, where η is an appropriate positive number satisfying $Z_\xi|_{\xi=\eta} \geq 0$.

Theorem 7.4. Assume $C = 0$ in QF and let (7.16) (7.20) be satisfied. Then basis B of (7.9) appears at most once as a feasible basis in Step A2 of Algorithm A, provided that ξ_0 is set to the above η in Step A1.

Proof: In this case, B of (7.9) is given by

$$B = \begin{bmatrix} -\xi D & A_1' & \mathbf{0} \\ A_1 & \mathbf{0} & \mathbf{0} \\ A_2 & \mathbf{0} & I_{m-k} \end{bmatrix} .$$

In a manner similar to the proof of Lemma 7.1, we have

$$\Delta \ (= \det B) = \det H(\xi) \det [-A_1 H(\xi)^{-1} A_1']$$

$$= (-\xi)^{n-k} \det D \det [-A_1 D^{-1} A_1'] .$$

Thus B^{-1} in this case has the following form.

$$B^{-1} : \begin{array}{c} \underbrace{}_{n} \quad \underbrace{}_{k} \quad \underbrace{}_{m-k} \\ \left[\begin{array}{c:c:c} \alpha/\xi & \alpha & \alpha \\ \hdashline \alpha & \xi\alpha & \alpha \\ \hdashline \alpha/\xi & \alpha & \alpha \end{array}\right] \begin{array}{l} \} \ n \\ \} \ k \\ \} \ m-k \end{array} \end{array} ,$$

where ψ stands for a real number independent of ξ (each ψ may represent a different number). Since it can be re-written as

$$\begin{pmatrix} -c(\xi) \\ b \end{pmatrix} : \begin{pmatrix} \xi\alpha+\beta \\ \hdashline \alpha \end{pmatrix} \begin{matrix} \} \ n \\ \} \ m \end{matrix}$$

by using the same notation as above, we have

$$\begin{bmatrix} x_B \\ u_B \\ y_B \end{bmatrix} = B^{-1} \begin{bmatrix} -c(\xi) \\ \\ b \end{bmatrix} : \begin{bmatrix} \frac{\xi\alpha+\beta}{\xi} \\ \hline \xi\alpha+\beta \\ \hline \frac{\xi\alpha+\beta}{\xi} \end{bmatrix} \begin{matrix} \} \ n \\ \\ \} \ k \\ \\ \} \ m-k \end{matrix}$$

This proves that the numerator of each element of x_B, u_B, y_B is linear in ξ; the denominator does not change its sign by Lemma 7.1 and $\xi > 0$. Thus we can let $r_i = 1$ in (7.15), obtaining $N(k) = 1$ in the proof of Theorem 3.3. □

7.6 Dinkelbach Method and Its Modification

In order to solve QF, the Dinkelbach method [Dinkelbach,'67] may be directly applied. It obtains $\bar{\xi}^*(\geqq 0)$ and the corresponding feasible solution $\bar{x}^*$ of QF such that

$$0 \leqq Z_{\bar{\xi}^*} \leqq \varepsilon,$$

where ε is a given nonnegative constant.

Dinkelbach's algorithm

Step D1: $i \leftarrow 0$, $\xi_i \leftarrow 0$ (or an appropriate values $Z_{\xi_0} \geqq 0$).

Step D2: Solve $Q'(\xi_i)$. If $Q'(\xi_i)$ is infeasible, so is QF; halt. Otherwise let x-part of a feasible solution of $Q'(\xi_i)$ be x^{ξ_i} (i.e., x^{ξ_i} is an optimal solution of $Q'(\xi_i)$).

Step D3: If $Z_{\xi_i} \leqq \varepsilon$, let $\bar{\xi}^* \leftarrow \xi_i$ $\bar{x}^* \leftarrow x^{\xi_i}$ and halt; otherwise

$$\xi_{i+1} \leftarrow (\frac{1}{2}\, x^{\xi_i\prime} C x^{\xi_i} + r' x^{\xi_i} + s)/(\frac{1}{2} x^{\xi_i\prime} D x^{\xi_i} + p' x^{\xi_i} + q)$$

$$i \leftarrow i+1$$

and return to D2.

If $\varepsilon > 0$, this algorithm halts after executing Step D2 and D3 finitely many times. If $\varepsilon = 0$, i.e., an exact optimal solution is sought, however, this algorithm usually requires an infinite number of iterations of Step D2 and D3. This difficulty can be easily removed by modifying it as follows, by making use of the property that interval

$[\xi_i, \xi_i']$ maintaining the same basis is easily calculated as discussed in Section 7.3. (This idea is also implicitly used in the numerical example in Appendix of [Dinkelbach,'67].)

Algorithm B (Modified Dinkelbach method)

Step B1: $i \leftarrow 0$, $\xi_i \leftarrow 0$ (or an appropriate value such that $Z_{\xi_0} \geqq 0$).

Step B2: Solve $Q'(\xi_i)$. If $Q'(\xi_i)$ is infeasible, so is QF; halt. Otherwise obtain the maximum ξ_i' such that $\xi_i' \geqq \xi_i$ and $Q'(\xi)$ has the same feasible basis for all $\xi \in [\xi_i, \xi_i']$. Let x-part of a feasible solution of $Q'(\xi_i')$ be $\hat{x}^i$ (i.e., $\hat{x}^i$ is an optimal solution of $Q'(\xi_i')$).

Step B3: If $Z_{\xi_i'} \geqq 0$ or $\xi_i' = \infty$, go to Step B4; otherwise

$$\xi_{i+1} \leftarrow \left(\frac{1}{2}\hat{x}^{i\prime} C\hat{x}^i + r'\hat{x}^i + s\right)/\left(\frac{1}{2}\hat{x}^{i\prime} D\hat{x}^i + p'\hat{x}^i + q\right)$$

$$i \leftarrow i+1$$

and return to Step B2.

Step B4: Calculate $\xi^* \in [\xi_i, \xi_i']$ such that $Z_{\xi^*} = 0$ and halt; ξ^* is the maximum value of QF.

The computational process of these algorithms are illustrated in Figure 7.2, in which solid arrows correspond to Algorithm B and broken arrows to the Dinkelbach method. ξ_i generated in Dinkelbach method is shown with bar $\bar{\xi}_i$ to distinguish it from ξ_i of Algorithm B. It is noted that ξ_{i+1} obtained in Step D3 (respectively Step B3) is given

as the intersection point of ξ-axis and the line tangent to Z_ξ at $\bar{\xi}_i$ (respectively ξ_i'). From Figure 7.1, it may be seen that Algorithm B requires less number of iterations than the original Dinkelbach method.

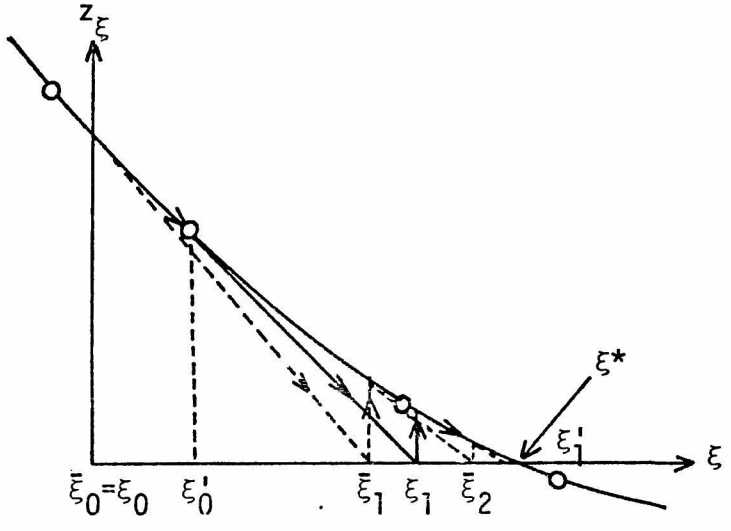


Figure 7.2. Illustration of computational processes of the Dinkelbach method and Algorithm B. (Solid arrows indicate Algorithm B and broken arrows indicate the Dinkelbach method.)

Remark 7.5. $Q'(\xi_i)$ $(i > 0)$ in Step D2 or Step B2 may be solved starting from the final tableau of $Q'(\xi_{i-1})$. This is a great computational saving, compared with solving

$Q'(\xi_i)$ from scratch. However, it may still require a considerable number of pivot operations if ξ_i is far from ξ_{i-1}. This should be compared with the fact that only one pivot operation is required in Algorithm A to solve $Q'(\xi_i)$ (Remark 7.1). Although Step B2 of Algorithm B is usually carried out far fewer times than Step A2 of Algorithm A, it is not clear which of Algorithm A and Algorithm B is more efficient.

Theorem 7.5. Algorithm B gives an optimal solution of QF or indicates its infeasibility, after executing Step B2 finite times, provided that $Q'(\xi_i)$ is solved by a finite algorithm.

Proof: The validity of Algorithm B follows from the validity of the Dinkelbach method [Dinkelbach,'67]. The finiteness can be proved in a manner similar to Algorithm A (see Theorem 7.2).

7.7 Computational Results

Algorithm A and Algorithm B discussed in the previous sections are implemented in *FORTRAN* and run on *FACOM 230/60* (roughly equivalent to *IBM 360/65* or *UNIVAC 1108*) and *FACOM 230/75* (roughly equivalent to *IBM 370/165*) of Kyoto University. Problems QF with $D = 0$ (see (7.1)) are exclusively treated in the experiment.

For various sizes n and m of QF, coefficients are randomly generated by the following rule:

C: A negative definite symmetric martix C of size n×n is obtained by

$$C = -PP'$$

for a nonsingular matrix P. P is generated by (1) randomly specifying nonzero elements of P with probability *NZC* (program parameter), (2) assigning a nonnegative (two digit) number randomly taken from the uniform distribution with interval [0.0, *9.9*] to each nonzero element, and (3) randomly inverting the sign of nonzero elements with probability *NC* (program parameter). Note that C has a considerably higher density than *NZC*; for example, C of two typical problems in Table 7.1. have nonzero densities *89.5%* for $NZC = 0.4$ and *56%* for $NZC = 0.25$.

A: An m×n matrix A is generated by (1) randomly specifying nonzero elements with probability *NZA* (program parameter), (2) assigning a nonnegative element, and (3) randomly inverting the sign of nonzero elements with probability *NA* (program parameter).

Table 7.1. Computational results for problems with n=20 and m=20.
(All figures are the average of 10 problems.)

Algorithm	$Q'(\xi_0)$		$Q'(\xi_i)$, $i>0$		Iteration (e)	Total time (sec.) (c, f)	Remark
	Pivot(a)	Time(c) (sec.)	Pivot(b)	Time(c) (sec.)			
A	114.7	12.7	14.1	1.9	15.1	14.6	*NZA*=0.4-0.7 *NZC*=0.2-0.5
B	114.7	12.7	19.9	2.9	2.6	15.6	*NA* =0.2-0.4 *NC* =0.2-0.4

r, p: Each element is assigned a (two digit) non-negative number randomly taken from [0.0, 9.9].

b: Each element is assigned a (four digit) positive number randomly taken from [100.0, 199.9].

$s = 20.0$ and $q = 3.0$ for all problems.

Table 7.1 and Table 7.2 summarize the computational results. The results in Table 7.1 are the average of 10 problems with n=m=20, while Table 7.2 lists results for each problem of larger size.

From these results, it may be concluded that Algorithm A is slightly faster than Algorithm B but there is no significant difference. It is also noticed that computation for $Q'(\xi_0)$ (the first one) in Step A2 or Step B2 is rather expensive compared with the rest (i.e., computation for $Q'(\xi_i)$, $i > 0$); computation for $Q'(\xi_0)$ consumes roughly 80∿90% of the entire computational time. Thus the use of parametric programming technique seems quite effective. This also explains a reason for the similarity (in computational time) of Algorithm A and Algorithm B mentioned above; two algorithms differ only in the way of generating ξ_i for $i > 0$.

It is also observed that program parameters specifying the ratio of nonzero and negative coefficients do not have much influence on the relative behavior of Algorithm A and B, though the higher values tend to increase the computational time. Table 7.1 includes problems with various parameter values.

In conlusion, it can be said that the quadratic fractional programming problem with $D = 0$ is a rather easy non-

Table 7.2. Computational results for large problems

Problem	Algorithm	$Q'(\xi_0)$		$Q'(\xi_i)$, $i>0$		Iteration (e)	Total time (sec.) (d, f)	Remark
		Pivot (a)	Time(d) (sec.)	Pivot (b)	Time(d) (sec.)			
n=40	A	344	31.8	25	2.7	25	34.5	*NZA*=0.6
m=40	B	344	31.8	22	2.5	2	34.3	*NZC*=0.4
								NA=*NC*=0.2
n=50	A	586	85.3	25	4.2	26	89.5	*NZA*=0.7
m=50	B	586	85.3	53	9.6	2	94.9	*NZC*=0.4
								NA=*NC*=0.2
n=50	A	606	83.2	27	4.5	28	87.7	*NZA*=0.5
m=50	B	606	83.2	68	12.5	2	95.7	*NZC*=0.4
								NA=*NC*=0.2
n=50	A	575	71.1	24	3.9	25	75.0	*NZA*=0.35
m=50	B	575	71.1	39	6.7	2	77.8	*NZC*=0.4
								NA=*NC*=0.2
n=50	A	694	110.6	29	4.8	30	115.4	*NZA*=0.4
m=50	B	694	110.6	57	9.5	3	120.1	*NZC*=0.17
								NA=*NC*=0.2

Notes

(a) The number of pivots required to solve $Q'(\xi_0)$ by Wolf's method.

(b) The number of pivots required to solve $Q'(\xi_i)$ for all $i > 0$.

(c) Machine is *FACOM 230/60*.

(d) Machine is *FACOM 230/75*.

(e) The number of executions of A2 (or B2) including the ones for $Q'(\xi_0)$.

(f) Including the computation for A3, A4 (or B3, B4).

linear programming problem, which can be solved in computational effort only slightly greater than that required for the well known (concave) quadratic programming problem.

7.8 Conclusion

The quadratic fractional programming problem is not always easy to deal with, in spite of its similarity to the well known quadratic programming. This chapter has shown that this problem has some analogy with discrete fractional programming problems discussed in Chapters 5 and 6 from the view point of solution procedures, and proposed two algorithms. One is a straightforward application of the parametric programming techinique of quadratic programming for finding ξ^* satisfying $Z_{\xi^*}=0$, and the other is a modification of Dinkelbach's approach. Both algorithms utilizes the Kuhn-Tucker condition for $Q(\xi)$ considered over the basic matrix of $Q'(\xi)$ to make next ξ_i as large as possible.

As shown in Section 7.7, the followings have been observed : (i) The parametric programming approach is slightly faster than the Dinkelbach's, but there is no significant difference. (ii) The quadratic fractional programming problem can be usually solved in only slightly greater computational time (about $10\% \sim 20\%$) than that required by the ordinary (concave) quadratic programming problems.

As pointed out in the previous discussion the zero point ξ^* of Z_ξ, i.e., ξ^* such that $Z_{\xi^*}=0$, plays a crucical role in solving QF. Algorithm B makes use of the local shape of Z_ξ, therefore, it may also be possible to use powerful numerical techniques developed for searching the zero point of a function. The incorporation of such a

technique, e.g., Newton Method, into general type of fractional programming problems may help develop the efficient algorithms. This direction should be further explored.

CHAPTER 8 CONCLUSION

In this dissertation, we have discussed a number of discrete programming problems. Specifically, new solution algorithms were proposed for these problems. Here, we summarize the obtained results and indicate some directions of future research.

In Chapter 2, we have proposed a version of the ordinary assignment problem which is defined on a PERT type network, and shown that the problem can be treated by branch-and-bound algorithm with partial problems. Each partial problem can be solved by Dantzig-Wolfe type decomposition procedure using subroutines of the ordinary assignment problem and the critical path problem.

In addition to this network model, many variants of the assignment problem may be conceivable. However, these variations are apt to lose the nice property that LP relaxation solves the original problem which is observed in the ordinary assignment problem. This property has been a source of various efficient algorithms for the assignment problem. Therefore variants having this property, if any, are worth investigating and seems to permit efficient algorithms. Fractional assignment problem may be candidate for a problem in such a class.

The constraint represented by a network may be interesting in itself. It arises also in other discrete programming problems in conjunction with the constraint imposed by some technological ordering conditions and/or resource conditions. Scheduling and sequencing problems

are some typical examples in this category. Further investigation of the structure and the solution methods of these problems from the view point of discrete programming seems to be fruitful in both theoretical and practical sense.

In Chapter 3, a new variant of the traveling salesman problem was introduced. It differs from the ordinary one in that it has a minimax type objective function. Although the direct application of dynamic programming approach does not seem to be possible, this approach becomes possible when the original problem is decomposed into (n-1) subproblems. Our algorithm may take $O(2^{n-3} \times n^3)$ time in the worst case, although only a very small fraction of this time is usually required in practice, due to several properties of the problem useful in improving the efficiency. (Of course, the problem itself is a difficult one in the theoretical sense discussed later). One aspect which was not fully exploited in this dissertation is the use of a lower bound for each subproblem P_δ in a branch-and-bound procedure. It can sometimes exclude some unpromising subproblems from the consideration without actually solving them [Morin,T.C., and R.E. Marsten,'76]. For example, the optimal value of the corresponding assignment problem with some additional constraints may be used for this purpose. This line of improvement should be further investigated. The minimax type objective function itself may deserve further exploration. For example, some type of scheduling problem with a minimax objective function is important in practice,

and allows an efficient algorithm [Kameda,'77]. The bottleneck objective function studied in conjunction with various problems ([Gross,O.,'58] [Shapiro,D.,'66]) may also be considered as a special case of minimax type objective function. It would be necessary to examine whether some techniques developed for the bottleneck problems can be extended to the minimax problems.

In Chapter 4, we have introduced the capacity constraint into the minimax type traveling salesman problem and ordinary traveling salesman problem. The original problem was decomposed into (n-1) subproblems which are slightly different from the subproblems discussed in Chapter 3 and dynamic programming approach was also developed. Although all hamiltonian circuits in a complete directed graph correspond to feasible solutions of the ordinary traveling salesman problem, only the hamiltonian paths along which the capacity constraint is satisfied are feasible in our problem. In this sense our problem is a constrained traveling salesman problem, which has not been studied so far. Investigation of other constrained traveling salesman problems may also be interesting.

Chapter 5-7 deal with fractional programming problems. In Chapter 5, we have treated the integer fractional programming problem. It was shown that the primal cutting plane algorithm called Young's SPA, for the ordinary integer programming problem can be easily generalized to handle this type of problem. However, a straightforward application of the technique of Chapter 5 with Glover's SPA seems to be difficult. Considering that Glover's SPA

may be favorably compared with Young's SPA from the view point of computational efficiency [Salkin,'75], it would be a subject for future research to find an algorithm making use of Glover's SPA.

As shown in Chapter 5, the integer programming and the integer fractional programming are closely related in nature. Therefore, many other techniques of the former may be extensible to the latter. Our algorithm in Chapter 5 should be regarded as only a starting example in this direction of research.

In Chapter 6, we have discussed the fractional knapsack problem. It asks to obtain an integer solution that maximizes its linear fractional objective function under the costraint of one linear inequality. A modification of Dinkelbach's algorithm was proposed to exploit the fact that good feasible solutions are easily obtained for both the fractional knapsack problem and the ordinary knapsack problem. An upper bound of the number of iterations required by this algorithm was derived. It was also clarified how optimal solutions depend on the right hand side of the constraint.

Our technique that uses a good feasible solution of each subsidiary problem $P(\xi)$ may be extensible to other types of fractional programming problems. There could be some other useful properties inherent in the fractional objective function. In this respect, the duality theory such as the one discussed in [Shaible,'76a and '76b] may be used to construct more efficient algorithms for fractional programming problems. Research on general proper-

ties of fractional programming problems should thus be further pursued.

In Chapter 7, we discussed quadratic fractional programming problems, and proposed two algorithms: One is a straightforward application of the parametric programming technique of quadratic programming and the other is a modification of the Dinkelbach method. According to the computational experiment, we have seen that there is no significant difference between two algorithms, and the quadratic fractional programming problems can be usually be solved in computational time only slightly greater (about 10%∿20%) than that required by the ordinary (concave) quadratic programming problems.

Chapter 5-7 clarified the following advantages of fractional programming problems. These problems are non-convex but easy to handle compared with other non-convex programming problems. Moreover many other problems can be considered as special cases of fractional programming problems (e.g., the fractional knapsack problem, the fractional assignment problem, etc). On the other hand, certain fractional programming problems becomes subproblems of other problems, e.g., maximum probability model of stochastic linear programming. In addition to the above mentioned theoretical importance, fractional programming problems have the practical importance. The marine transportation problem ([Pollack, et.al.,'65] [Bitran and Novaes,'73]) is an example of such application of fractional programming as is stated in Section 5.1.

As a final remark of this chapter, some recent results

on the theory of computational complexity are briefly mentioned. As also noted in the previous chapters, many of known discrete programming problems such as the traveling salesman problem and the knapsack problem have long been considered to be computationally intractable. Despite the intensive efforts of many creative people, no efficient algorithms have been found for them, which are guaranteed to require computational time bounded by a polynomial function of the problem size. The brief in the inherent difficulty of these problems has been strengthened by the results of S.A.Cook [Cook,'71] and R.M.Karp [Karp,'72]. These show that the above problems, together with a wide variety of other discrete programming problems, form a class of NP-complete problems. No member in this class is known to have a polynomial time algorithm. Moreover the theory shows that all NP-complete problems have polynomial algorithms if at least one of them has such an algorithm. This would be most unlikely. Since it is easy to show that discrete programming problems considered in this dissertation are all NP-complete, no problem seems to have a polynomial time algorithm as far as the worst case behavior is concerned. Therefore, the best we can expect is to look for algorithms which may run in exponential time in the worst case but works efficiently on the most problems practically encountered. In reality, it is worthwhile to have algorithms which are faster than previous ones even though they require exponential time in the worst case, thereby extending the computational limit of the maximum size of problems which can

be practically handled by a computer. Our efforts in this dissertation have always be directed in this direction.

The other alternative, which is commonly taken, is to find fast (heuristic) algorithms which are guaranteed to yield approximate solutions that are close to exact optimal solutions. Although these heuristic algorithms may not actually find exact optimal solutions, the obtained solutions would be practically almost as useful as exact optimal solutions if they are close enough to the optimal. No effort was made in this dissertation to explore the possibility of this direction. However, there is no doubt that this would be one of the most fruitful and important direction in the field of discrete programming. Extension and modification of our results along this line should be urgently pursued.

REFERENCES

Agin,Y. (1966), "Optimal Seeking with Branch-and-Bound", *Management Science* 13B pp176-185.

Anzai,Y. (1974), "On integer fractional programming", *Journal of Operations Research Society of Japan* 17 pp49-66.

Aris,R. (1964), *Discrete Dynamic Programming*, Braisdell Publishing Company, New York.

Balas,E. (1965), "An Additive Algorithm for Solving Linear Programs with Zero-One Variables", *Operations Research* 13 pp517-546.

Balas,E. (1967), "Discrete Programming by the filter method", *Operations Research* 15 pp915-957.

Balas,E. (1968), "A Note on the Branch-and-Bound Principle", *Operations Research* 16 pp442-445.
Errata. *Operations Research* 16 pp886.

Balas,E. (1971), "Intersection Cuts - A New Type of Cutting Plane for Integer Programming", *Operations Research* 19 pp19-39.

Bellman,R. (1957), *Dynamic Programming*, Princeton University Press, Princeton, New Jersey.

Bellman,R. (1962a), *Adaptive Control Processes: A Guided Tour*, Princeton University Press, Princeton, New Jersey.

Bellman,R. (1962b), "Dynamic Programming Treatment of the Traveling Salesman Problem", *J. Assn. for Computing Machinery* 9 pp61-63.

Bellman,R. and S.E.Dreyfus (1962). *Applied Dynamic Programming*, Princeton University Press, Princeton, New Jersey.

Berge,C. (1962), *The Theory of Graphs*; Matheden, London.

Berge,C. (1973), *Graphs and Hypergraphs*, North Holland Publishing Company : Transaction and revised edition of "*Graphes et Hypergraphs*" Dunod Paris, 1970.

Bertier,P., and B.Roy (1964), "Procédure de Résolution pour une classe de Problems Pouvant Avoir un Caractère Combinatoire", *Cashiers Cent d'Etudes Recherche Operationelle* 6 pp202-208: Translated by W.Jewell, *ORC Rept.* pp17-34, University of California, Berkeley, 1967.

Bitran,G.R., and A.G.Novaes (1973), "Linear Programming with a Fractional Objective Function", *Operations Research* 21 pp22-29

Cabot,V.A. (1970), An Enumeration Algorithm for Knapsack Problems", *Operations Research* 18 pp306-311.

Charnes,A., and W.W.Cooper (1962), "Programming with linear fractional functionals", *Naval Research Logistics Quarterly* 9 pp181-196.

Charnes,A., and W.W.Cooper (1963), Deterministic Equivalents for Optimizing and Satisficing Under Chance Constraints", *Operations Research* 11 pp18-39.

Charnes,A., W.W.Cooper, R.J.Niehaus and A.Stedry (1969) "Static and dynamic assignment models with multiple objectives and some remarks on organization design", *Management Sceince* 15B pp365-375.

Christofides,N., (1975), *Graph Theory : An Algorithmic Approach*, Academic Press, New York.

Dantzig,G.B., (1957), "Discrete Variable Extreme Problems", *Operations Research* 5 pp266-277.

Dantzig,G.B., (1960), "On the Significance of Solving Linear Programming Problems with Some Integer Variables", *Econometrica* 28 pp30-40.

Dantzig,G.B., and P.Wolfe (1960), "Decomposition Principle for Linear Programming", *Operations Research* 8 ppl01-lll.

Dinkelbach,W. (1967), "On nonlinear fractional programming", *Management Sceince* 13 pp492-498.

Elmaghraby,S.E., and S.Arisawa (1972), "On hyperbolic programming with a single constraint and upper-bounded variables", *Management Sceince* 19 pp42-45.

Ford,L.R.Jr., and D.R.Fulkerson (1962), *Flows in Networks*, Princeton University Press, Princeton, New Jersey.

Garfinkel,R.S., and G.L.Nemhauser (1972), *Integer Programming*, John Wiley & Sons, New York, N.Y.

Geoffrion, A.M. (1967a), "Integer Programming by Inplicit Enumeration and Balas' Method", *SIAM. Rev.* 7 ppl78-190.

Geoffrion,A.M. (1967b), "Stochastic Programming with Aspiration and Fractile Criterion", *Management Science* 13 pp672-679.

Geoffrion, A.M. (1970), "Elements of large-scale mathematical programming", *Management Science* 16 pp652-691.

Gilmore,P.C. (1962), "Optimal and Sub-optimal Algorithms for the Quadratic Assignment Problems", *SIAM. J.* 10 pp305-313.

Gilmore,P.C., and R.E.Gomory (1961), "A Linear Programming Approach to the Cutting Stock Problem", *Operations Research* 9 pp849-859.

Gilmore,P.C., and R.E.Gomory (1963), "A Linear Programming Approach to the Cutting Stock Problem, Part II", *Operations Research* 11 pp863-888.

Gilmore,P.C., and R.E.Gomory (1964), "Sequencing a State-Variable Machines", A Solvable Case of the Traveling Salesman Problem", *Operations Research* 12 pp655-679.

Gilmore,P.C., and R.E.Gomory (1965), "Multistage Cutting Stock Problems of Two and More Dimensions", *Operations Research* 13 pp94-120.

Gilmore,P.C., and R.E.Gomory (1966), "The Theory and Computation of Knapsack Functions", *Operations Research* 14 pp1045-1074.

Glover,F. (1965), "A Multiphase - Dual Algorithm for the Zero-One Integer Programming Problem", *Operations Research* 13 pp879-919.

Glover,F. (1968), "A New Foundation for a Simplified Primal Integer Programming Algorithm", *Operations Research* 16 pp727-740.

Glover,F. (1969), "*Convexity Cuts*", Graduate School of Business University of Texas.

Glover,F. (1973), "Convexity Cuts and Cut Search", *Operations Research* 21 pp123-134.

Golomb,S.W. and L.D. Baumert (1965), "Backtrack Programming", *J. Assn. for Computing Machinery* 12 pp516-524.

Gomory,R.E. (1958), "Outline of an Algorithm for Integer Solutions to Linear Programs", *Bull. Amer. Math. Soc.* 64 pp275-278.

Gomory,R.E., (1965), "On the Relation between Integer and Non-Integer Solutions to Linear Program", *Proc. Nat. Acad. Sci.* 53 pp260-265.

Gorry,G.A., and J.F. Shapiro (1971), "An Adaptive Group Theoretic Algorithm for Integer Programming Problem", *Management Science* 17 pp285-306.

Greenberg,H. (1969), "An Algorithm for the Computation of Knapsack Functions", *J. Math. Anal. and Appl.* 26 ppl59-162.

Greenberg,H. and R.L.Hegerich (1970), "A Branch Search Algorithm for the Knapsack Problem", *Management Science* 16 pp327-332.

Gross,O. (1958), "The Bottleneck Assignment Problem", *Rand Corporation Paper,* P-1630 March 6.

Gruspan, M. (1973), "Hyperbolic integer programming", *Naval Research Logistics Quarterly* 20 pp341-356.

Hadley,G. (1964), *Nonlinear and Dynamic Programming* Addison Wesley.

Hammer,P.L. and S.Rudeanu (1968), *Boolean Methods in Operations Research and Related Areas,* Springer-Verlag, Berlin.

Harary,F. (1969), *Graph Theory,* Addison Wesley Publishing Company, Menlo Park, California.

Hart,P.E., N.J.Nilsson and B. Raphael (1968), "A formal basis for the heuristic determination of minimal cost paths, "*IEEE Trans. System Science and Cypernetics*, SSC-4 pp100-107.

Held,M. and R.M.Karp (1962), "A Dynamic Programming Approach to Sequencing Problems", *J. Soc. Indust. Appl. Math.* 10 pp196-210.

Held,M. and R.M.Karp (1970), "The Traveling-Salesman Problem and Minimum Spanning Trees: Part II", *Mathematical Programming* 1 pp6-25.

Howard,R.A. (1960), *Dynamic Programming and Markov Processes*, John-Wiley & Sons, New York.

Hu,T.C., (1969), *Integer Programming and Netwok Flows*, Addison-Wesley.

Hu,T.C. and M.L. Lenard (1973), "*A study of a heuristic algorithm*", MRC. Tech. Rept. No.1370, The University of Wisconsin-Madison Mathematics Research Center, Madison, Wisconsin.

Ibaraki,T. (1973), "Algorithms for Obtaining Shortest paths Visiting Specified Nodes", *SIAM. Rev.* 15 pp309-317.

Ibaraki,T. (1976a), "The computational efficiency of approximate branch-and bound algorithm", *Mathematics of Operations Research* 1 pp287-298.

Ibaraki,T. (1976b), "Theoretical comparisons of search strategies in branch-and-bound algorithms", *International Journal of Computer and Information Sciences*, 5 pp315-344.

Ibaraki,T. (1977), "On the computational efficiency of branch-and-bound algorithms", *Journal of Operations Research Society of Japan* 20 pp16-35.

Ibaraki,T. (1977), "The power of dominance relations in branch-and-bound algorithm", *J. Assn. for Computing Machinery* 24 pp264-279.

Ibaraki,T., H.Ishii, and H.Mine (1976a), "An Assignment Problem On a Network", *Journal of Operations Research Society of Japan* 19 pp70-90.

Ibaraki,T., H.Ishii, J.Iwase, T.Hasegawa, and H.Mine (1976b), "Algorithms for Quadratic Fractional Programming Problems", *Journal of Operations Research Society of Japan* 19 pp174-191.

Ishii,H., T.Ibaraki, and H.Mine (1975a), " ミニマックス型目標関数をもつある巡回セールスマン問題 " (in Japanese), Keiei Kagaku 19 pp112-122.

Ishii,H., T.Ibaraki, and H.Mine (1975b), "Traveling Salesman Problem with Capacity Constraint of the delivery Truck", *The Memoirs of the Faculty of Engineering, Kyoto University* 37 pp252-263.

Ishii,H., T.Ibaraki, and H.Mine (1976), "A Primal Cutting Plane Algorithm for Fractional Integer Programming Problem", *Journal of Operations Research Society of Japan* 19 pp228-244.

Ishii,H., T.Ibaraki, and H.Mine (1977) "Fractional Knapsack Problems", *Mathematical Programming* 13 pp255-271.

Ishii,H., T.Nishida, and Y.Nanbu (1978), "A Generalized Chance Constraint Programming Problem, *Journal of Operations Research Society of Japan* 21 pp124-145.

Jagannathan,R. (1966), "On Some properties of programming problems in parametric form pertaining to fractional programming", *Management Science* 12 pp609-615.

Jeroslow,R.G. (1974), "The Principles of Cutting-Plane Theory Part I", *Management Science Research Report No.332*, Carnegie-Melon University.

Karp,R.M. (1972), "Reducibility Among Combinatorial Problems", in Complexity of Computer Computations, R.E.Miller and J.W.Thatcher, eds., Plenum Press, New York pp85-104.

Kolesar,P.J. (1967), "A Branch-and-Bound Algorithm for the Knapsack Problem", *Management Science* 13 pp723-735.

Kuhn,H.W. (1955), "The Hungarian method for the assignment problem", *Naval Research Logistics Quarterly* 2 pp83-97.

Kuratowski, K. (1930), "Sur le problème des coirbes gauches en topologie", *Fund. Math.* 15 pp275-283.

Lawler,E.L. (1963), "The Quadratic Assignment Problem", *Management Science* 9 pp586-599.

Lowler,E.L., and D.E.Wood (1966), "Branch-and Bound Methods: A Survey", *Operations Research* 14 pp699-719.

Little,J.D.C., K.G.Murty, W.Sweeney, and C. Karel (1963), "An Algorithm for the Traveling Salesman Problem", *Operations Research* 11 pp979-989.

Magazine,M.J., G.L.Nemhauser and L.E.Trotter (1975), "When the greedy solution solves a class of knapsack problems", *Operations Research* 23 pp203-217.

Martos,B. (1964), "Hyperbolic programming", *Naval Research Logistics Quarterly* 11 ppl35-155.

Mitsumori,S. (1974) Research Report of Automatic Control Group AC 74-13, Institute of Electrical Engineering of Japan.

Mitten,L.G. (1970), "Branch-and-Bound Methods: General Formulation and Properties", *Operations Research* 18 pp24-34.

Morin,T.L., and R.E.Marsten (1976), "Branch-and-Bound Strategies for Dynamic Programming", *Operations Research* 24 pp611-627.

Munkres,J. (1957), "Algorithms for the assignment and transportation problem", *SIAM.J.* 5 pp32-38.

Nemhauser,G.L. (1966), *Introduction to Dynamic Progrmming*, John Wiley & Sons.

Nilsson,N.J. (1971), *Problem-Solving Methods in Artificial Intelligence*, McGraw-Hill, New York.

Pierskalla,W.P. (1968), "The Multidimensional assignment problem", *Operations Research* 16 pp422-431.

Pollack,E.G., G.N.Novaes and E.G.Frankel (1965), "Optimization and Integration of Shipping Ventures", *International Shipbuilding Progress* pp267-281.

Ritter,K. (1962), "Ein Verfahren zur Lösung parametrabhängiger, nichitlinear Maximum-Probleme", *Unternehmensforschung* 6 ppl40-166; English transration by M.Meyer, "A method for solving nonlinear maximum-problems depending on parameters", *Naval Research Logistics Quarterly* 14 (1967) ppl47-162.

Robillard,P. (1971), "(0,1) Hyperbolic Programming", *Naval Research Logistics Quarterly* 18 pp47-58.

Sahni,S. (1975), "Approximate algorithms for the 0/1 knapsack problem", *J. Assn. for Computing Machinery* 22 ppll5-124.

Salkin,H. (1972), "A Note Comparing Glover's and Young's Simplified Primal Algorithms", *Naval Research Logistics Quarterly* 19 pp399-402.

Salkin, H. (1975), *Integer Programming*, Addison Wiseley, Menlo Park California.

Salkin,H., P.Schroff and S.Mehta (1974), "Primal All-Integer Integer Programming Applied to Tableaux with Rational Coefficients", *Technical Memorandum* No.298. Department of Operations Research, Case Western Researve University.

Schaible,S. (1976a), "Fractional Programming. I, Duality", *Management Sceince* 22 pp858-867.

Schaible,S. (1976b), "Fractional Programming.II, On Dinkelbach's Algorithm", *Management Sceince* 22 pp868-873.

Shapiro,D. (1966), "*Algorithms for the Solution of the Optimal Cost and Bottleneck Traveling Salesman Problem*", Doctoral Thesis, School of Engineering and

Applied Science, Washington University, St. Louis, Missouri.

Shapiro,J.F. (1968a), "Dynamic Programming Algorithms for the Integer Programming Problem-I: The Integer Programming Problem Viewed as a Knapsack Type Problem", *Operations Research* 16 pp103-121.

Shapiro,J.F. (1968b),. "Group Theoretic Algorithms for the Integer Programming problem-II : Extension to a General Algorithm", *Operations Research* 16. pp928-947.

Shapiro,J.F., and H.M.Wagner (1967), "A Finite Renewal Algorithm for a Knapsack and Turnpike Models", *Operations Research* 15 pp319-341.

Shiode,S., H.Ishii, and T.Nishida (1978), "Stochastic Transportation Problems", *Technology Reports of the Osaka University* 27 pp1-8.

Silver,R. (1960), "Algorithm 27: Assignment", *Communications of ACM* 3 pp603-604.

Stancu-Minasian I.M. (1977), "Bibliography of Fractional Programming 1960-1976", Reprint No.3 Academy of Economic Studies Department of Economic Cybernetics.

Thomas,M.E. (1976), "A Survey of the State of the Art in Dynamic Programming", *American Institute of Industrial Engineering Transaction* 8 pp59-69.

Tutte,W.T. (1965), "Lectures on Matroid", *J.Research National Bureau of Standards B* 69 pp1-48.

Wald,A. (1950), *Statistical Decision Functions*, John-Wiley & Sons, New York.

White,W.W. (1966), *"On a Group Theoretic Approach to Linear Integer Programming"*, ORC 66-27, Operations Research Center, University of California, Berkeley.

Whitney,H. (1935), "On the abstract properties of Linear Dependence", *Am. J. of Math.* 57 pp509-533.

Wolfe,P. (1959), "The Simplex method for quadratic programming", *Econometrica* 27 pp382-298.

Young,R.D. (1965), "A Primal (all-Integer) Integer Programming Algorithm", *J. Res. Nat. Bur. Stads. 69B* pp213-250.

Young,R.D. (1968), "A Simplified Primal (all-Integer) Integer Programming Algorithm", *Operations Research* 16 pp750-782.